THE MINIMUM COST FLOW PROBLEM: PRIMAL ALGORITHMS AND COST PERTURBATIONS

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

by

P. THIRUGNANA SOKKALINGAM

to the

DEPARTMENT OF MATHEMATICS
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

SEPTEMBER, 1995

CERTIFICATE

This is to certify that the matter embodied in the thesis entitled **"THE MINIMUM COST FLOW PROBLEM: PRIMAL ALGORITHMS AND COST PERTURBATIONS"** by Mr P. THIRUGNANA SOKKALINGAM for the award of Degree of Doctor of Philosophy of the Indian Institute of Technology Kanpur is a record of bonafide research work carried out by him under our supervision. The results embodied in this thesis have not been submitted to any other University or Institute for award of any Degree or Diploma.

(Dr. Ravindra K. Ahuja)
Associate Professor
Department of Industrial
and Management Engineering
I.I.T. Kanpur - 208 016, India

(Dr. Prabha Sharma)
Professor
Department of Mathematics
I.I.T. Kanpur - 208 016, India

September '95

SYNOPSIS

The minimum cost flow problem has been receiving continuous intensive research for last five decades due to its wide applications in areas such as operations research, applied mathematics, computer science and other related areas. In this dissertation, our contributions to this area of research are two fold : (i) developing new primal algorithms, (ii) studying the effects of cost perturbations.

We first study the primal network simplex algorithm with a new entering arc rule, which we call as the ***minimum ratio pivot rule***. This rule assigns a unit penal-ty to each arc violating optimal-ty conditions in the initial basis. Then every pivot iteration selects a nonbasic arc whose fundamental cycle has the minimum cost-to-penality ratio as the entering arc. The algorithm performs at most n^2 consecutive degenerate pivots and runs in pseudo-polynomial time. When applied to the shortest path problem and the assignment problem, the algorithm runs in $O(nm + n^2 \log n)$ time. Here n denotes the number of nodes and m denote the number of arcs in the given network.

We next develop two cycle cancelling algorithms. Our first cycle cancelling algorithm repeatedly selects an eligible arc (that is, violating its optimality conditions) with the largest residual capacity. Then either the selected arc is made noneligible, or its residual capacity is reduced to half of its previous capacity by augmenting flow along a negative cycle. When

arc capacities are integers, the algorithm runs in $O(m \log U(m + n \log n))$, where U denotes the largest capacity over all arcs.

Our second cycle cancelling algorithm performs the following steps repeatedly on the residual network. The subgraph of the residual graph consisting of eligible arcs is made acyclic. Then a cut with noneligible forward arcs and atleast one eligible backward arc is found; the maximum optimality violation over the arcs in this cut is brought down. When arc costs are integral, the algorithm runs in $O(nm \log n \log(nC))$, where C is the largest cost over all arcs. Incorporating a shortest path subroutine in the iterative procedure, we make the algorithm strongly polynomial. The strongly polynomial version of this algorithm runs in $O(nm^2(\log n)^2)$ time.

We next consider the arc tolerance analysis for the separable convex cost flow problem. Here we consider the problem of perturbing cost function associated with an arc while maintaining optimality of a given flow. For an arc (k, ℓ), we define arc *tolerance interval* $[\alpha_{k\ell}, \beta_{k\ell}]$ as the interval having the property that any cost function having at least one of its left and right derivative in the interval maintains the optimality of the given flow. We characterize $\alpha_{k\ell}$ and $\beta_{k\ell}$ in terms of nonsingleton shortest distances between node k and node ℓ in the residual network. For the minimum cost flow problem, this is the interval in which the cost co-efficient $c_{k\ell}$ can vary while maintaining the optimality of the given flow. We give an $O(n^3)$ algorithm to compute tolerance intervals for all arcs in the network.

We next consider inverse network flow problems. Given an observed flow and a priori estimated cost c, the inverse network flow problem is to perturb c to a "nearest" cost t for which the

given flow becomes a minimum cost flow. We call such a vector t as a ***conormal*** at the given flow and we measure the "nearness" by a ***variance function*** $k(t-c)$, where k is the function having all properties of a norm except for symmetry. Using duality results in convex analysis, we develop the theory for duality in the general context where the observed solution is a solution to a system of linear constraints and the "nearness" is measured by a general variance function $k(t-c)$. We show the following results for inverse network flow problems.

(i) The dual of Euclidean inverse network flow problem is equivalent to a separable strictly convex quadratic cost flow problem. The optimal solutions of the primal and dual problems give a decomposition of the given cost c.

(ii) The dual of weighted rectilinear inverse network flow problem is a minimum cost flow problem. A minimum cost disjoint augmenting cycles with respect to the given flow is an optimum solution to the dual of the rectilinear inverse network flow problem.

(iii) The dual of the weighted maximum inverse network flow problem is equivalent to finding a minimum cost-to-weight ratio augmenting cycle with respect to the given flow.

(iv) The dual of the weighted rectilinear inverse spanning tree problem is transformed to a transportation problem.

We also construct an optimal solution to the corresponding inverse network flow problems from the information obtained in solving the respective dual problems.

Dedicated to

My Parents, My Teachers
and
My Maternal Uncle

ACKNOWLEDGEMENT

I have great pleasure in expressing my sincere gratitude to my thesis supervisors, Dr. Prabha Sharma and Dr. Ravindra K. Ahuja, for excellent guidance, illuminating discussions, constructive criticism and constant encouragement throughout the course of this study. Their understanding nature, love and patience are highly appreciated. Through my association with them, I have benefited both academically as well as personally.

I take this opportunity to thank Dr. S.K. Gupta and Dr. A.K. Mittal for their suggestions and help to my thesis work. I am grateful to Dr. P. Shunmugaraj for many useful discussions and timely helps. My sincere thanks to Drs. P.C. Das, V. Raghavendra, G.K. Shukla, S. Madan, D. Kundu, Santokh Singh, Sadagopan, A.P. Sinha and other teaching staffs who have helped and encouraged me in many ways. It is a pleasure to acknowledge my colleagues Guha, Rajesh, Durai, Siva, Rao, Panda, Amit, Padma, Pratibha, Alphonse, my IME colleagues - Sivakumar, Neeraj, Kutubuddin, and many others for their help and company during my stay in the Campus.

I express my sincere thanks to Dr. A.P. Punnen for many helps that I received from him. I am grateful to Drs. Adi Ben-Israel, S. Pallatino, S.N. Kabadi, R. Chandrasekaran, H. Narayanan and L.S. Thakur for sharing their valuable time and experience with me. I also thank Drs. J.B. Orlin and U. Derigs for providing references to me.

I thank Mr. Ghanshyam Hoshing for the meticulous typing/ drawing of my thesis work and for the company provided during this period. My special thanks to my friends Ponds, Seenu, Rajesh, Palani, and Venki who have helped me to take the final print-out of my thesis.

I thank all non-teaching staff of Maths and IME departments for their help. My special thanks to Sharmaji for his homeopathic treatment and 'Canteen' Narayanan for his valuable service.

Many friends have made my stay at IIT Kanpur a memorable one. It will be difficult to forget friends like Kasi, Ilango, Ponds, 'kadi' Siva, Durai, Seenu, "RSel", Manoravi, Govind, Raghunathan, Subbu, Ramesh, Tamil, Sampath, "Aero" Elango, Periya Pandiyan, Narayanan, Dharmaraj, Haja, Yogi, Thaliyati, Shivde, "Aruvaz" Pandiyan, Kumar, Saravanan, Arumugam, Sivaguru, Guru Prem, Perips, Venki, Guru, Jeebu, Palanivel, Justin, Mohanraj, Renga, "rules" Siva, Bhatia, Raghavan, TIFR Ravi, Kadilkar, Karandikar, Sayan Kar, Vinod, "G.K." Deepak, Joydeep, Dada, "Chess Club", Medical Sundar and Govind, Army and Airforce friends, Scientist, NSI Venky, kotees, Bucket, Pancha Pandavas (Rajesh, Ravi, Palani, Raghu, Uma), Vivek, Nagarajan, Sivakumar, Ramesh Chandran, Ravichandran, Ramesh Rao, Marthandam, Johnson, Uricha Kozhi, and many others who do not figure in this list.

I also thank Mrs. Smita Ahuja, Mrs. Chitra Manoravi, Mrs. Andal Ramesh, Mrs "RSel", Mrs. Usha Elango, Mrs. Dharmaraj and Mrs. Nivedita Rajesh for their hospitality.

I am grateful to all my school and college teachers who have encouraged me throughout my studies. My special thanks to Mr. Natarajan, Mr. Thangamuthu and their family members for their support during my graduation.

I am grateful to my relative Mr. Balasubramani and his family members for the support during my graduation.

I am indebted to my parents, my maternal uncle and aunt, my sister and my immediate relatives for their constant support and encouragement.

P. THIRUGNANA SOKKALINGAM

CONTENTS

Page No.

Chapter 1. **INTRODUCTION**

1.1 Introduction 1

1.2 Preliminaries 4

1.2.1 Graph Theoretical Notations and Definitions 4

1.2.2 Minimum Cost Flow Problem 7

1.2.3 Complexity Analysis 11

1.3 Outline of the Thesis 14

Chapter 2. **NETWORK SIMPLEX ALGORITHMS FOR THE MINIMUM COST FLOW PROBLEM**

2.1 Introduction 20

2.2 The Network Simplex Algorithm 26

2.3 The Minimum Ratio Pivot Rule 36

2.4 Bounding the Number of Pivots 39

2.5 Bounding the Running Time 44

2.6 Specific Implementation 51

CHAPTER 3. **CYCLE-CANCELLING ALGORITHMS FOR THE MINIMUM COST FLOW PROBLEM**

3.1 Introduction 53

3.2 The Background 56

3.3 A Cycle-cancelling Algorithm with Capacity Scaling 60

3.4 A Cycle-cancelling Algorithm by Sharing Optimality Violation 66

CHAPTER 4. **ARC TOLERANCES IN NETWORK FLOW PROBLEMS**

4.1 Introduction 82

4.2 The Background and the Problem Formulation 87

4.2.1 Optimality Conditions 91

4.2.2 Arc Tolerance Analysis 95

4.3 Theory 96

4.4 Algorithm for Finding Non-singleton Shortest Paths 100

CHAPTER 5. **INVERSE LINEAR PROGRAMMING AND NETWORK FLOW PROBLEMS**

5.1 Introduction 114

5.2 Preliminaries 117

5.3 Inverse Linear Programming Problem 121

5.4 Inverse Network Flow Problems 136

5.4.1 Euclidean Variance 140

5.4.2 Weighted Rectilinear Variance 143

5.4.3 Weighted Maximum Variance 150

5.5 Inverse Spanning Tree Problem 156

5.5.1 Weighted Rectilinear Variance 160

CHAPTER 6. **CONCLUDING REMARKS** 163

REFERENCES 166

Chapter 1

INTRODUCTION

1.1 INTRODUCTION

Highways, telephone lines, electric power systems, computer chips, water delivery system, and rail lines: these physical networks and many others, are familiar to all of us. In each of these problem settings, the minimum cost flow problem arises naturally and plays a pivotal role. The problem is that we wish to send some good(s) (vehicles, messages, electricity, or water) from source(s) to destination(s) in a network at minimum possible cost. Networks are also used to represent logical relationships in important practical problems. As a result, the minimum cost flow problem solves a wide variety of applications arising in facility location, production planning, project management, operations scheduling and cash management, apart from obvious transportation and communication applications in physical networks (see Ahuja et al. [1993] and references therein).

Researchers have proposed many algorithms to solve the minimum cost flow problem. Most of the fundamental algorithms such as the primal network simplex algorithm, the primal-dual algorithm, the successive shortest path algorithm, the out-of-

kilter algorithm and the cycle-canceling algorithm were developed in 1950s and 1960s. The development of the first polynomial algorithm by Edmonds and Karp [1972] and the first strongly polynomial algorithm by Tardos [1985] have created interest in developing polynomial and strongly polynomial versions of traditional algorithms. Researchers also studied theoretical behaviour such as degeneracy, cycling, and stalling in the primal network simplex algorithm due to its theoretical and practical importance.

Perturbations of arc costs that preserve optimality is a well-known problem under post-optimality analysis. The so-called basis preserving cost operator theory by Srinivasan and Thompson [1972] and tolerance analysis by Shier and Witzgall [1980] consider this issue for an optimal basis (spanning tree structure) obtained by a primal network simplex algorithm. Recently, Burton and Toint [1992, 1994] have studied cost perturbations that make a given solution optimal in the context of shortest paths. Their problem, called as inverse shortest path problem, is to perturb arc costs such that given set of paths become shortest paths after perturbations, and the perturbation is minimum with respect to the Euclidean norm.

In this dissertation, we investigate the following algorithms and problems:

i) The primal network simplex algorithm with a new entering arc rule.

ii) Two variants of cycle-canceling algorithms.

iii) Arc tolerances with respect to an optimal flow for the more general separable convex cost flow problem.

iv) The inverse network flow problem.

v) The inverse spanning tree problem.

The proposed primal network simplex algorithm assigns unit penalty to each arc violating its optimality condition in the initial basis. At each iteration, an arc with the minimum cost-to-penalty ratio is selected as an entering arc. We analyze the anti-stalling property and overall complexity in the worst-case sense. The complexity of the algorithm for the special cases, the shortest path and assignment problem is analyzed. We next propose two variants of the cycle-canceling algorithm. The first algorithm uses the idea of capacity scaling. The second algorithm reduces the maximum optimality violation in a cut at least by a factor of two. The second variant becomes strongly polynomial when it is enhanced by a shortest path subroutine.

We next study arc tolerances with respect to an optimal flow instead of an optimal basis. We develop the theory for the more general separable convex cost flow model by properly defining arc tolerances. We characterize arc tolerances by nonsingleton shortest paths between the head and tail nodes of the corresponding arc. We then develop an algorithm to compute tolerance intervals for all arcs which essentially computes nonsingleton shortest paths between all pair of nodes.

We next study the inverse network flow problem. The problem is to perturb a priori estimated set of arc costs such that a given feasible flow becomes a minimum cost flow and the perturbation is minimum with respect to a norm like function. We first derive the dual of inverse linear programming problem from

the duality results for convex programming. We show that duals of the inverse network flow problems in which perturbations are measured by the Euclidean, the weighted rectilinear, and the weighted maximum variances are, respectively, equivalent to solving standard problems, namely, the separable strictly convex quadratic flow problem, the minimum cost circulation problem, and the minimum cost-to-weight ratio problem. We also show that the dual of the weighted rectilinear inverse flow problem is a transportation problem.

1.2 PRELIMINARIES

In this section we give basic definitions from graph theory and complexity theory and formulations of the minimum cost flow problems and other related formulations.

1.2.1 Graph Theoretical Notations and Definitions

A *directed graph* $G = (N, A)$ consists of a set N of nodes and a set A of arcs whose elements are ordered pairs of distinct nodes. A *directed network* is a directed graph whose nodes and/or arcs have associated numerical values (typically, costs, capacities, and/or supplies and demands). We often make no distinction between graphs and networks, so we use the terms "graph" and "networks" synonymously. We let n denote the number of nodes and m denote the number of arcs in G. An *undirected* graph/network is defined in the same way as a directed graph/ network except that arcs are unordered pairs of distinct nodes. A directed arc (i, j) has two *endpoints* i and j. We refer to node i as the *tail* of the arc (i, j) and node j as its *head*. We say that the arc (i, j) *emanates* from node i and *terminates* at node j. An

arc (i, j) is ***incident*** to nodes i and j. An arc (i, j) is an ***outgoing arc*** of node i and an ***incoming arc*** of node j. The ***forward star*** $F(i)$ of node i is the set of arcs emanating from node i and the ***backward star*** $B(i)$ is the set of arcs terminating at node i. The forward star $F(i)$ and the backward star $B(i)$ are also used to refer to the head nodes of outgoing arcs at node i and the tail nodes of incoming arcs at node j. We let $A(i)$ denote the set of all outgoing and incoming arcs at node i. Note that $\sum_{i \in N} |F(i)| = \sum_{i \in N} |B(i)| = m$, and $\sum_{i \in N} |A(i)| = 2m$. We assume that no arc has the same node as its head and tail, and no two or more arcs have the same tail and head nodes. That is, there are no ***loops*** and ***multiarcs***. A graph $G' = (N', A')$ is a ***subgraph*** of $G = (N, A)$ if $N' \subseteq N$ and $A' \subseteq A$.

A ***walk*** in a directed graph $G = (N, A)$ is a subgraph of G consisting of a sequence of nodes and arcs i_1-a_1-i_2-a_2- ... i_{r-1}-a_{r-1}-i_r satisfying the property that for all $1 \le k \le r-1$, either $a_k = (i_k, i_{k+1}) \in A$ or $a_k = (i_{k+1}, i_k) \in A$. We shall sometimes omit arcs or nodes in the representation of a walk. A directed walk is a walk in which every two consecutive nodes i_k and i_{k+1} are connected by the arc (i_k, i_{k+1}). A ***path*** is a walk without any repetition of nodes. A cycle is a path i_1-i_2-...-i_r together with the arc (i_r, i_1) or (i_1, i_r). We shall often refer to a cycle using the notation i_1-i_2- ... -i_r-i_1. An arc (i, j) in a path/cycle is a ***forward arc*** if the path/cycle visits node i priori to visiting node j, and is a ***backward arc*** otherwise. The set of forward arcs and the set of backward arcs of a path P (cycle W) are denoted respectively by $\overline{P}(\overline{W})$ and $\underline{P}(\underline{W})$. We say orientation of a cycle is same as that of an arc (i, j) in the

cycle if (i, j) is a forward arc in the cycle, and is opposite to that of arc (i, j) if it is a backward arc in the cycle. A ***directed path/cycle*** is a path/cycle in which every arc is a forward arc.

A graph is acyclic if it contains no directed cycle. We will say that two nodes i and j are ***connected*** if the graph contains at least one path from node i to node j. A graph is connected if every pair of its nodes is connected; otherwise, the graph is disconnected. We assume that the graphs considered in this thesis are connected.

A ***cut*** is a partition of the node set N into two parts, S and $\bar{S} = N-S$, where S is a non-empty proper subset of N. Each cut defines a set of arcs consisting of those arcs that have one end point in S and another point in $\bar{S}$. Therefore, we refer to this set of arcs as a cut and represent it by the notation $[S, \bar{S}]$. An arc (i, j) is a ***forward*** arc of the cut $[S, \bar{S}]$ if $i \in S$ and $j \in \bar{S}$, and is a ***backward arc*** if $i \in \bar{S}$ and $j \in S$. We denote the set of forward arcs in $[S, \bar{S}]$ by $(S, \bar{S})$, and the set of backward arcs by $(\bar{S}, S)$. An ***s-t cut*** is defined with respect to two distinguished nodes s and t and is a cut $[S, \bar{S}]$ satisfying the property that $s \in S$ and $t \in \bar{S}$.

A ***tree*** is a connected graph that contains no cycle. A connected subgraph of a tree is a subtree. A ***rooted tree*** is a tree with a specially designated node, called its ***root***; we regard a rooted tree as though it were hanging from its root. A tree is a ***directed out-tree rooted at node s*** if the unique path in the tree from node s to every other node is a directed path.

A tree T is a *spanning tree* of G if T contains all nodes of G. We refer to the arc belonging to a spanning tree T as *tree* arcs and arcs not belonging to T as *nontree* arcs. The addition of any nontree arc to the spanning tree T creates exactly one cycle. We refer to any such cycle as a *fundamental cycle* of G with respect to the tree T. Since a basis of the minimum cost flow problem corresponds to a spanning tree T, tree arcs, nontree arcs and fundamental cycles are also alternatively referred to as basic arcs, nonbasic arcs and basic cycles respectively. The deletion of any tree arc (i, j) of the spanning tree T produces a disconnected graph containing two subtrees T_i containing node i and T_j containing node j. This also produces an i-j cut whose node partition is given by nodes in T_i and T_j. We sometimes denote this cut by $[S_i, S_j]$ where S_i and S_j are respectively the nodes spanning the trees T_i and T_j.

A graph $G = (N, A)$ is a *bipartite* graph if we can partition its node set into two subsets N_1 and N_2 so that for each arc (i, j) in A either (i) $i \in N_1$ and $j \in N_2$, or (ii) $i \in N_2$ and $j \in N_1$.

1.2.2 Minimum Cost Flow Problem

We now present a mathematical programming formulation of the minimum cost flow problem (*MCF*) and then describe several of its specializations and variants as well as the separable convex cost flow problem that we consider in this thesis.

Let $G = (N, A)$ be a directed network defined by a set N of n nodes and a set A of m directed arcs. Each arc $(i, j) \in A$ has an associated *cost* c_{ij} that denotes the cost per unit flow on that arc. We assume that the flow cost varies linearly with the amount

of flow. We also associate with each arc $(i, j) \in A$ a ***capacity*** u_{ij} (also called as ***upper bound***) that denotes the maximum amount that can flow on the arc and a ***lower bound*** ℓ_{ij} that denotes the minimum amount that must flow on the arc. We associate with each node $i \in N$ an integer number $b(i)$ representing its ***supply/demand***. If $b(i) > 0$, node i is a ***supply node***; if $b(i) < 0$, node i is a ***demand node***; if $b(i) = 0$, node i is a ***transshipment node***. The decision variables in the minimum cost flow problem are ***arc flows*** and we represent the flow on arc $(i, j) \in A$ by x_{ij}. The minimum cost flow problem is an optimization model formulated as follows:

$$\text{Minimize} \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{1.1a}$$

subject to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i) \text{ for all } i \in N, \tag{1.1b}$$

$$\ell_{ij} \le x_{ij} \le u_{ij} \qquad \text{for all } (i, j) \in A, \tag{1.1c}$$

where $\sum_{i=1}^{n} b(i) = 0$. In matrix form, we represent the minimum cost flow problem as follows :

$$\text{Minimize } cx \tag{1.2a}$$

subject to

$$Nx = b, \tag{1.2b}$$

$$\ell \le x \le u. \tag{1.2c}$$

In this formulation, N is an nxm matrix, called the ***node-arc incidence matrix*** of the minimum cost flow problem. Each column N_{ij} in the matrix corresponds to the variable x_{ij}. The column N_{ij} has a +1 in the i^{th} row, a -1 in the j^{th} row; the rest of its entries are zero.

We refer to the constraints in (1.1b) as ***flow conservation equations***. The flow must also satisfy the lower bound and capacity constraints (1.1c), which we refer to as ***flow bound constraints***. We refer to both (1.1b) and (1.1c) as the system of ***flow constraints on G*** and denote it by F. We often assume all lower bounds are zero; if we do not state lower bounds for any problem, we assume that they have value zero.

We consider the following special cases of the minimum cost flow problem in the thesis.

Shortest path problem : The problem is to find a path of minimum cost (or length) from a specified ***source node s*** to another specified ***sink node t***, assuming that each arc $(i, j) \in A$ has an associated cost (or length) c_{ij}. If we set $b(s) = 1$, $b(t) = -1$, and $b(i) = 0$ for all other nodes in the minimum cost flow problem, the solution to the problem will send 1 unit of flow from node s to node t along shortest path. If we want to determine shortest paths from source node s to every other node in the network, then in the minimum cost flow problem we set $b(s) = (n-1)$ and $b(i) = -1$ for all other nodes. [We can set each arc capacity u_{ij} to any number larger than (n-1)]. The minimum cost flow solution would then send unit flow from node s to every other node i along a shortest path. Both versions of the problem essentially find a directed out-tree rooted at node s in which the tree path from node s to every other node is a shortest path; we refer to this tree as a ***shortest path tree from node s***. In linear programming terminology, this corresponds to an optimum basic solution of the problem.

Assignment problem : The data of the assignment problem consist of two equally sized sets N_1 and N_2 (i.e., $|N_1| = |N_2|$), a collection of pairs $A \subseteq N_1 \times N_2$ representing possible alignments, and a cost c_{ij} associated with each element $(i, j) \in A$. In the assignment problem we wish to pair, at minimum possible cost, each object in N_1 with exactly one object in N_2. The assignment problem is a minimum cost flow problem in a network $G = (N_1 \cup N_2, A)$ with $b(i) = 1$ for all $i \in N_1$, $b(i) = -1$ for all $i \in N_2$, and $u_{ij} = 1$ for $(i, j) \in A$.

Transportation problem : The transportation problem is a special case of the minimum cost flow problem with the property that the node set N is partitioned into two subsets N_1 and N_2 (of possibly unequal cardinality) so that (1) each node in N_1 is a supply node, (2) each node N_2 is a demand node, and (3) for each arc (i, j) in A, $i \in N_1$ and $j \in N_2$.

Circulation problem : The circulation problem is a minimum cost flow problem with only transshipment nodes; that is, $b(i) = 0$ for all $i \in N$. In this instance, we wish to find a feasible flow that honours the lower and upper bounds ℓ_{ij} and u_{ij} imposed on the arc flow x_{ij}. Since we never introduce any exogenous flow into the network or extract any flow from it, all the flow circulates around the network. We wish to find the circulation that has the minimum cost.

In this thesis, the separable convex cost flow problem (*SCCF*), a generalization of the minimum cost flow is considered for arc tolerance analysis.

Separable convex cost flow problem : In this model, the cost of an arc flow is a convex function of the amount of flow on the arc. That is, the objective (1.1a) is replaced by the following convex cost function :

$$C(x) = \sum_{(i,j)\in A} C_{ij}(x_{ij}), \tag{1.3}$$

where $C_{ij}(x_{ij})$ is a real-valued convex function of the amount of flow x_{ij}.

An inverse problem of the following minimum spanning tree problem is considered in the thesis.

Minimum spanning tree problem : The cost of a spanning tree of an undirected graph is the sum of the costs (or lengths) of its arcs. In the minimum spanning tree problem, we wish to identify a spanning tree of minimum cost (or length).

1.2.3 Complexity Analysis

We measure the performance of algorithms in the worst-case sense. *Worst-case analysis* aims to provide upper bounds on the number of steps that a given algorithm can take on any problem instance. For a worst-case analysis, we bound the running time of network algorithms in terms of several basic problem parameters, the number n of nodes, the number m of arcs, and upper bounds C and U on the cost coefficients and the arc capacities. Whenever C (or U) appears in the complexity analysis, we assume that each cost (or capacity) is integer valued. To express the running times, we rely extensively on $O(\ )$ notations. For example, we write that Dijkstra's shortest path algorithm runs in $O(m+n \log n)$ time. This means that if we count the number of arithmetic steps

taken by the algorithm, then the number of steps taken by any instance $\leq c(m+n \log n)$ for some constant c; the constant c does not depend on the parameters of the problem.

An algorithm is said to be a ***polynomial time*** algorithm if its running time is bounded by a polynomial function of the input length. The input length of a problem is the number of bits needed to represent it. For a network problem, the input length is a low-order polynomial function of n, m, log C and log U. Consequently, a network algorithm is a polynomial time algorithm if its running time is bounded by a polynomial function of n, m, log C and log U. A polynomial time algorithm is said to be a ***strongly polynomial time algorithm*** if its running time is bounded by a polynomial function in only n and m. We say that an algorithm is a ***pseudo-polynomial time algorithm*** if its running time is polynomially bounded in n, m, C, and U.

The study of network flow models predates the development of linear programming. The first studies in this domain, conducted by Kantorovich [1939], Hitchcock [1941], and Koopmans [1947], considered the transportation problem, a special case of the minimum cost flow problem. These studies provided insight into the problem structure and yielded algorithmic approaches. Interest in the network flow problems grew with the advent of the simplex method by Dantzig in 1947, who also specialized this algorithm for the transportation problem (see Dantzig [1951]).

During the 1950s, researchers began to exhibit increasing interest in the minimum cost flow problem and its specializations mainly because of the importance of these models in real-world

applications. Dantzig, Ford, and Fulkerson pioneered the development of special algorithms for solving these problems. Whereas Dantzig focused on the simplex based methods, Ford and Fulkerson developed primal-dual combinatorial algorithms. The landmark books by Dantzig [1962] and Ford and Fulkerson [1962] present thorough discussion of these early contributions.

The following are some important algorithmic approaches developed for solving the minimum cost flow problem.

1. Primal network simplex algorithm (Dantzig [1951, 1962]).
2. Primal-dual algorithm (Ford and Fulkerson [1957, 1962]).
3. Successive shortest path algorithm (Jewell [1958], Iri [1960], Busaker and Gowen [1961]).
4. Out-of-kilter algorithm (Minty [1960], Fulkerson [1961]).
5. Cycle-cancelling algorithm (Klein [1967]).
6. Relaxation algorithm (Bertsekas and Tseng [1988]).

Primal network simplex algorithm and Relaxation algorithm are found to be efficient in practice.

The capacity scaling algorithm by Edmonds and Karp [1972] pioneered the development of polynomial algorithms for the minimum cost flow problem. Rock [1980] and, independently, Bland and Jensen [1985] suggested a cost scaling technique. The capacity scaling and the cost scaling techniques are further refined by Orlin [1984] and Goldberg and Tarjan [1987]. The first strongly polynomial time algorithm for the minimum cost flow problem is proposed by Tardos [1985]. Her idea of "fixing arc flows" and Orlin [1984, 1988] idea of "fixing node potentials" played important role in the development of strongly polynomial

algorithms. Fujishige [1986], Galil and Tardos [1986], Goldberg and Tarjan [1988], Erlovina and McCormick [1990], and Gabow and Tarjan [1989] made contributions in this field. Orlin [1984] and Plotkin and Tardos [1990] have developed polynomial-time dual network simplex algorithms. Very recently, Orlin [1995] settled the question of developing polynomial primal network simplex algorithm positively by developing one such algorithm.

The best available time bound for the minimum cost flow problem is $O(\min \{nm \log(n^2/m) \log(nC), nm (\log\log U) \log (nC), m \log n (m+n \log n)\}$. These bounds are obtained, respectively, by Goldberg and Tarjan [1987], Ahuja, Goldberg, Orlin, and Tarjan [1992], and Orlin [1988].

The bibliographies on network optimization prepared by Golden and Magnanti [1977], Ahuja, Magnanti, and Orlin [1989, 1991], Bazaraa, Jarvis, and Sherali [1990], and the extensive set of references on integer programming compiled by researchers at the University of Bonn (Kartning [1976], Hausman [1978], and Von Randow [1982, 1985]) provide good references on the developments in network optimization. Some survey papers on applications of network optimization are Bennington [1974], Glover and Klingman [1977], Glover, Klingman and Phillips [1990], and Ahuja, Magnanti, Orlin and Reddy [1995]. The book by Ahuja, Magnanti and Orlin [1993], and the survey paper by Ahuja et al. [1995] describe more than 100 applications and provide additional references to many more applications.

1.3 OUTLINE OF THE THESIS

In Chapter 2, we present a variant of the primal network simplex algorithm, with a new entering arc rule, which we call as

the *minimum ratio pivot rule*. This rule assigns unit penalty to each arc violating optimality conditions in the initial basis. We define the penalty of a fundamental cycle created by a nonbasic arc in the same way as the cost is defined. Then every pivot iteration selects a nonbasic eligible arc whose fundamental cycle has the minimum cost-to-penalty ratio. We maintain a strongly feasible basis (Cunningham [1976]) throughout the execution of the algorithm. Cunningham [1979] and Goldfard, Hao and Kai [1990b] proposed several antistalling pivot rules for the network simplex algorithm which prevent exponential number of consecutive degenerate pivots. Our rule is also an antistalling pivot rule which allows at most n^2 consecutive degenerate pivots. Though our algorithm is pseudo-polynomial, our algorithm runs in $O(nm + n^2 \log n)$ time for the shortest path problem and the assignment problem; this complexity matches the complexity of best implementations of the network simplex algorithms for these problems developed by Glover and Klingman [1988], and Akgul [1985, 1994]. Our implementation and analysis is simpler.

In Chapter 3, we develop two variants of the cycle-cancelling algorithm. The cycle-cancelling algorithm is credited to Klein [1967]. Goldberg and Tarjan [1988], Barhona and Tardon [1989] and Wallacher and Zimmerman [1991] have developed polynomial and strongly polynomial variants of this algorithm. Our first cycle-cancelling algorithm uses capacity scaling. Edmonds and Karp [1972] and Orlin [1988] have employed capacity scaling in their algorithm; their algorithms are not primal algorithms. Our algorithm selects an eligible arc with the maximum residual capacity. By solving a shortest path problem, either the selected

algorithms. Fujishige [1986], Galil and Tardos [1986], Goldberg and Tarjan [1988], Erlovina and McCormick [1990], and Gabow and Tarjan [1989] made contributions in this field. Orlin [1984] and Plotkin and Tardos [1990] have developed polynomial-time dual network simplex algorithms. Very recently, Orlin [1995] settled the question of developing polynomial primal network simplex algorithm positively by developing one such algorithm.

The best available time bound for the minimum cost flow problem is $O(\min \{nm \log(n^2/m) \log(nC), nm (\log\log U) \log (nC), m \log n (m+n \log n)\}$. These bounds are obtained, respectively, by Goldberg and Tarjan [1987], Ahuja, Goldberg, Orlin, and Tarjan [1992], and Orlin [1988].

The bibliographies on network optimization prepared by Golden and Magnanti [1977], Ahuja, Magnanti, and Orlin [1989, 1991], Bazaraa, Jarvis, and Sherali [1990], and the extensive set of references on integer programming compiled by researchers at the University of Bonn (Kartning [1976], Hausman [1978], and Von Randow [1982, 1985]) provide good references on the developments in network optimization. Some survey papers on applications of network optimization are Bennington [1974], Glover and Klingman [1977], Glover, Klingman and Phillips [1990], and Ahuja, Magnanti, Orlin and Reddy [1995]. The book by Ahuja, Magnanti and Orlin [1993], and the survey paper by Ahuja et al. [1995] describe more than 100 applications and provide additional references to many more applications.

1.3 OUTLINE OF THE THESIS

In Chapter 2, we present a variant of the primal network simplex algorithm, with a new entering arc rule, which we call as

the *minimum ratio pivot rule*. This rule assigns unit penalty to each arc violating optimality conditions in the initial basis. We define the penalty of a fundamental cycle created by a nonbasic arc in the same way as the cost is defined. Then every pivot iteration selects a nonbasic eligible arc whose fundamental cycle has the minimum cost-to-penalty ratio. We maintain a strongly feasible basis (Cunningham [1976]) throughout the execution of the algorithm. Cunningham [1979] and Goldfard, Hao and Kai [1990b] proposed several antistalling pivot rules for the network simplex algorithm which prevent exponential number of consecutive degenerate pivots. Our rule is also an antistalling pivot rule which allows at most n^2 consecutive degenerate pivots. Though our algorithm is pseudo-polynomial, our algorithm runs in $O(nm + n^2 \log n)$ time for the shortest path problem and the assignment problem; this complexity matches the complexity of best implementations of the network simplex algorithms for these problems developed by Glover and Klingman [1988], and Akgul [1985, 1994]. Our implementation and analysis is simpler.

In Chapter 3, we develop two variants of the cycle-cancelling algorithm. The cycle-cancelling algorithm is credited to Klein [1967]. Goldberg and Tarjan [1988], Barhona and Tardon [1989] and Wallacher and Zimmerman [1991] have developed polynomial and strongly polynomial variants of this algorithm. Our first cycle-cancelling algorithm uses capacity scaling. Edmonds and Karp [1972] and Orlin [1988] have employed capacity scaling in their algorithm; their algorithms are not primal algorithms. Our algorithm selects an eligible arc with the maximum residual capacity. By solving a shortest path problem, either the selected

arc is made noneligible, or its residual capacity is reduced at least by a factor of two by augmenting the flow along a negative cycle. When arc capacities are integers, the algorithm runs in $O(m \log U (m+n \log n)$ time.

Our second cycle-cancelling algorithm performs the following steps repeatedly on the residual network. The subgraph of the residual graph consisting of eligible arcs is made acyclic. This is done by using a cancel subroutine developed by Goldberg and Tarjan [1988]. Then a cut with noneligible forward arcs and at least one eligible backward arc is found; the maximum optimality violation over the arcs in this cut is reduced by a factor of at least two. When arc costs are integral, the algorithm runs in $O(nm \log n \log (nC))$ time. Incorporating a shortest path subroutine in the iterative procedure, the algorithm becomes strongly polynomial and runs in $O(nm^2(\log n)^2)$. The complexity of our algorithm matches the complexity of the cancel-and-tighten algorithm developed by Goldberg and Tarjan [1988].

In Chapter 4, we study arc tolerances for the separable convex cost flow problem which includes the minimum cost flow problem as its special case. Given an optimal flow, a convex cost function $\tilde{C}_{k\ell}(x_{k\ell})$ is a valid perturbation of the arc cost (convex) function $C_{k\ell}(x_{k\ell})$ if the replacement of the latter by the former preserves the optimality of the given flow. We show that there is an interval $[a_{k\ell}, \beta_{k\ell}]$, which we call as the ***tolerance interval*** of arc (k, ℓ), such that a convex function is a valid perturbation if and only if at least one of its marginal costs falls in the interval. Further, we characterize both $\alpha_{k\ell}$ and $\beta_{k\ell}$ in terms of nonsingleton shortest distances between nodes k and ℓ in the

residual network. For the minimum cost flow problem, this is the interval in which the cost coefficient $c_{k\ell}$ can vary while maintaining the optimality of the given flow. For the minimum cost flow problem, Shier and Witzgall [1980] have studied arc tolerances with respect to an optimal basis exploiting its spanning tree structure and have presented an $O(n^2)$ algorithm for computing arc tolerances for all arcs. Earlier, Srinivasan and Thompson [1972] have considered this issue for the transportation problem by so called basis preserving operators. The book by Ahuja et al. [1993] treats unit change in an arc cost combinatorially which can be extended for more general change. The book by Rockafellar [1984] contains a brief and general discussion on sensitivity analysis for the separable convex cost flow problem. Here our emphasis is on proper extension of arc tolerances to the convex cost flow problems and its characterization by nonsingleton shortest distances. We give an $O(n^3)$ algorithm for computing nonsingleton shortest distances between all pair of nodes. Our combinatorial approach enables us to study arc tolerances for any optimal solution to the minimum cost flow problem removing the effect of degeneracy as well as apply it to the more general convex cost flow problem.

In Chapter 5, we study inverse network flow problems. Given a priori estimated cost c, the "nearness" of an arbitrary cost t to c is measured by a variance function $k(t-c)$, where k is a function having all properties of a norm except for symmetry; we call such a function as an unorm. The inverse network flow problem is to perturb c to a cost t such that (i) a given observed flow is a minimum cost flow with respect to t, and (ii) $k(t-c)$ is

minimum over all such t's. Burton and Toint [1992, 1994] have recently studied the inverse shortest path problem with Euclidean variance. Their motivation for studying this problem is its application in urban transit model and computerized tomography. Their emphasis is on developing a algorithm using a formulation of constraints in terms of paths. In this dissertation, we study the duals of inverse network flow problems which include inverse shortest path problem if the given paths originate from a single source.

A natural generalization of the inverse network flow problem is the inverse linear programming problem where a given observed solution is a solution to a system of linear constraints. Using duality results in convex analysis, we first show that duals of inverse linear programming problems are certain feasible direction problems. Specializing this result, we show that duals of inverse network flow problems with Euclidean variance, the weighted rectilinear variance, and the weighted maximum variance are respectively equivalent to solving a separable strictly convex quadratic cost flow problem, a minimum cost flow problem and a minimum cost-to-weight ratio cycle problem. We also construct the required cost t after solving respective dual problems. We also show that minimum cost arc disjoint augmenting cycles with respect to the given flow represents an optimal solution to the rectilinear inverse network flow problem.

We finally show that the dual of the rectilinear inverse spanning tree problem is equivalent to a transportation problem. This demonstrates that the inverse problem defined on a type of combinatorial problem need not be of the same type.

Chapter 6 contains the concluding remarks.

Since the problem dealt in various chapters are quite different, we survey the relevant literature chapterwise.

Chapter 2

NETWORK SIMPLEX ALGORITHMS FOR THE MINIMUM COST FLOW PROBLEM

2.1 INTRODUCTION

The simplex algorithm, due to Dantzig [1951a], is one of the landmark contributions to computational mathematics of this century. The simplex algorithm plays a pivotal role in operations research and other related disciplines because of the pervasiveness of its applications throughout many problem domains, its extraordinary efficiency, and because it permits us to not only solve problems numerically, but also to gain considerable practical and theoretical insight through the use of sensitivity analysis and duality theory.

Minimum cost flow problems define a special class of well-structured linear programs. The network simplex algorithm is a special implementation of the general (primal) simplex algorithm to the minimum cost flow problem, which exploits the underlying network structure. The attractiveness and empirical efficiency of the algorithm is due to the fact that the basic solutions of a minimum cost flow problem correspond to the spanning trees of the network $G = (N, A)$ on which the problem is defined. Here N is the node set with $n = |N|$ nodes and A is the arc set with $m = |A|$ arcs. The essence of the algorithm is that it proceeds

successively from a spanning tree (solution) to another spanning tree (solution), each obtained from the previous one by adding a new edge and deleting a tree edge, until an optimal spanning tree (solution) satisfying the optimality conditions is found.

Dantzig [1951b] developed the network simplex algorithm for the uncapacitated transportation problem by specializing his linear programming simplex method. He proved the spanning tree property of the basis and the integrality property of the optimal solution. Later, his development of the upper bounding technique for linear programming led to an efficient specialization of the simplex method for the minimum cost flow problem. Dantzig's [1962] book discusses these topics.

The network simplex algorithm gained its current popularity in the early 1970s when the research community began to develop and test algorithms using efficient tree indices. Johnson [1966] suggested the first tree indices. Srinivasan and Thompson [1973], and Glover, Karney, Klingman, and Napier [1974] implemented these ideas; these investigations found the network simplex algorithm to be substantially faster than the existing codes that implemented the primal-dual and out-of-kilter algorithms.

Gassner [1964] has shown that the network simplex method can fail to terminate due to ***cycling***, cycling occurs when the algorithm encounters repeatedly the same sequence of trees. This happens only when the algorithm performs ***degenerate pivots*** which change the spanning tree but not the solution. The ***strongly feasible spanning tree technique***, proposed by Cunningham [1976] for the minimum cost flow problem, and independently by Barr,

Glover, and Klingman [1977] for the assignment problem, helps to reduce the number of degenerate pivots in practice and ensures that the network simplex algorithm has a finite termination. Although the strongly feasible spanning tree technique prevents cycling during a sequence of consecutive degenerate pivots, the number of consecutive degenerate pivots can be exponential. This phenomenon is known as ***stalling***. Cunningham [1979] and Goldfarb, Hao, and Kai [1990b] describe several antistalling pivot rules for the network simplex algorithm. The book by Ahuja et al. [1993, Chapter 11] develops the network simplex algorithm from first principles in a ***combinatorial*** way and deals with all the above concepts comprehensively. We adopt their approach for our algorithm. They also give a detailed literature survey on the network simplex algorithm.

Edmonds and Karp [1972] were the first to devise a polynomial-time algorithm for the minimum cost flow problem. This development led the researchers to investigate the polynomial-time network simplex algorithm for the problem among other types of algorithms.

Though many polynomial-time algorithms have been developed for the minimum cost flow problem (see, for example, Ahuja, Magnanti and Orlin [1993]), developing a polynomial-time network simplex algorithm for this problem has remained a challenging problem till very recently. Some previous partial success in this direction is a subexponential-time algorithm due to Tarjan [1991], and polynomial-time network simplex algorithms due to Goldfarb and Hao [1988] and Tarjan [1991] where they occasionally allow pivots to increase the objective function value. Very recently, Orlin

[1995] has succeeded in developing a *genuinely* polynomial-time network simplex algorithm for the minimum cost flow problem. However, many polynomial-time network simplex algorithms have been developed for the maximum flow problem, the assignment problem, and the shortest path problem.

In this chapter, we present a new primal simplex pivot rule, which we call the ***minimum ratio pivot rule***, and analyze the worst-case behavior of the resulting primal simplex algorithms for the minimum cost flow, assignment, and shortest path problems. We consider a directed network $G = (N, A)$ with node set N and arc set A. Let $n = |N|$ denote the number of nodes in the network, and $m = |A|$ denote the number of arcs in the network. Each arc $(i, j) \in A$ has a nonnegative integer arc capacity u_{ij}, and an arc cost c_{ij}. We denote by Δ any upper bound on the sum of all arc flows, i.e., $\sum_{(i,j)\in A} x_{ij} \leq \Delta$ in every feasible flow x.

Our primal simplex algorithm first defines a penalty d_{ij} for each arc (i, j) of the network, which is 0 or 1. With respect to the arc penalties d_{ij}'s and the arc costs c_{ij}'s, the minimum ratio pivot rule is to select that ***eligible arc*** as the entering arc whose addition to the basis creates a cycle with the minimum cost-to-penalty ratio. We show that the primal simplex algorithm with the minimum ratio pivot rule solves the minimum cost flow problem in $O(n\Delta)$ pivots. Using simple data structures, this algorithm can be implemented in $O(n^2\Delta)$ time, but using the Fibonacci heap data structure the running time can be improved to $O(\Delta(m + n \log n))$. For the assignment and shortest path problems, $\Delta = n$; hence the primal simplex algorithm with the minimum ratio

pivot rule solves these problems in $O(n^2)$ pivots and in $O(n(m + n \log n))$ time.

Srinivasan and Thompson [1977] used unit penalties in their area cost operator algorithm for the transportation problem. Since they perturb original cost, their algorithm is not genuinely primal simplex algorithm. Karp and Orlin [1981], and Young, Tarjan, and Orlin [1991] used unit penalties in their algorithms for special parametric analysis for the shortest path problem.

Table 2.1 presents a survey of the network simplex algorithms for the minimum cost flow problem and its special cases, and compares the worst-case complexity of the number of pivots and the time taken by these algorithms with the algorithms developed by us in this section. As is evident from Table 1, our network simplex algorithms for the minimum cost flow problem, the assignment problem, and the shortest path problem match the running times of the algorithms developed by Glover and Klingman [1988), and Akgul [1985, 1994]. Akgul's algorithms solve a sequence of problems on subgraphs $G_1 = (N, A_1)$, $G_2 = (N, A_2)$, ..., $G_r = (N, A_r)$, where $A_1 \subseteq A_2 \subseteq \ldots \subseteq A_r = A$. Given an optimal solution of the problem on G_K, the graph G_{K+1} is obtained by adding all arcs incident to a specific node, and reoptimization is done using the network simplex algorithm with Dantzig's pivot rule. The reoptimization routine bears a close resemblance to an application of Dijkstra's algorithm. Glover and Klingman's algorithm is also similar in spirit; it also solves a sequence of problems on subgraphs $G_1 = (N, A_1)$, $G_2 = (N, A_2), \ldots, G_w = (N, A_w)$, where G_{K+1} is obtained from G_K by adding all arcs in $(A - A_K)$ that satisfy optimality conditions and exactly one eligible arc in

Table 2.1 : Number of pivots and running times of various network simplex algorithms

Optimization Problems	Developers	Number of pivots	Running Time
Minimum Cost Flow Problem	Orlin [1985]	$O(n\Delta \log (mCU))$	$O(nm\Delta \log C)$
	Glover and Klingman [1988]	$O(n\Delta)$	$O(\Delta(m+n \log n))$
	Tarjan [1991]	$O(n^{(\log n/2 +1)}$	$O(n^{(\log n/2 +1)}$
		$m.\min \{\log(nC), m \log n\})$	$m^2\min\{\log(nC), m \log n\})$
	Ahuja and Orlin [1992]	$O(n\Delta \log C)$	$O(m\Delta \log C)$
	This chapter	$O(n\Delta)$	$O(\Delta(m+n \log n))$
	Orlin [1995]	$O(nm \min\{\log(nC), m \log n\})$	$O(n^2m \min\{\log(nC), m \log n\})$
Maximum Flow Problem	Goldfarb and Hao [1990]	$O(nm)$	$O(n^2m)$
	Goldberg, Grigoriadis & Tarjan	$O(nm)$	$O(nm \log n)$
Assignment Problem	Roohey-Laleh [1980]	$O(n^3)$	$O(n^3m)$
	Hung [1983]	$O(n^3 \log (nC))$	$O(n^3m \log nC)$
	Orlin [1985]	$O(\min\{n^2\log(nC), n^2m \log n\})$	$O(\min \{n^2m \log (nC), n^2m^2 \log n\})$
	Akgul [1985]	$O(n^2)$	$O(nm+n^2 \log n)$
	Glover and Klingman [1988]	$O(n^2)$	$O(nm+n^2 \log n)$
	Ahuja and Orlin [1992]	$O(n^2 \log C)$	$O(nm \log C)$
	This chapter	$O(n^2)$	$O(nm+n^2 \log n)$
Shortest Path Problem	Akgul [1985]	$O(n^2)$	$O(nm+n^2 \log n)$
	Orlin [1985]	$O(\min\{n^2\log (nC), n^2m \log n\})$	$O(\min\{n^2m \log (nC), n^2m^2\log n\})$
	Goldfarb, Hao and Kai [1990]	$O(n^2)$	$O(n^3)$
	Glover and Klingman [1988]	$O(n^2)$	$O(nm+n^2 \log n)$
	Ahuja and Orlin [1992]	$O(n^2 \log C)$	$O(nm \log C)$
	This chapter	$O(n^2)$	$O(nm+n^2 \log n)$

$(A - A_k)$ does not satisfy the optimality condition. Then, as in Akgul's algorithm, the network simplex algorithm with Dantzig's pivot rule is used for reoptimization. On the contrary, our algorithms consider all arcs in the network at the same time and use the eligible arc with the minimum cost-to-penalty ratio as the entering arc. We think that our pivot rule is quite natural, and its worst-case analysis is also considerably simpler.

The organization of this chapter is as follows.

In Section 2.2, we review the necessary background material for the minimum cost flow problem and present the network simplex algorithm in detail. In Section 2.3, we describe the minimum-ratio pivot rule in detail. In Section 2.4, we obtain a bound on the number of pivots performed by the resultant simplex algorithm. In Section 2.5, we obtain a worst-case time complexity of this simplex algorithm. In Section 2.6, we discuss the performance of the resultant algorithms for the special cases, the shortest path and assignment problems.

2.2 THE NETWORK SIMPLEX ALGORITHM

In this section, we present some background material for the minimum cost flow problem. The minimum cost flow problem has the following linear programming formulation:

$$\text{Minimize} \quad \sum_{(i,j)\in A} c_{ij}\, x_{ij} \tag{2.1a}$$

subject to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i) \quad \text{for all } i \in N, \tag{2.1b}$$

$$0 \le x_{ij} \le u_{ij} \quad \text{for all } (i, j) \in A. \tag{2.1c}$$

In this formulation, if $b(i) > 0$ then node i is called a ***supply node***, and if $b(i) < 0$ then node i is called a ***demand node***; if $b(i) = 0$, then node i is a transshipment node. We assume that all arc capacities u_{ij} are integer, all supplies/demands $b(i)$ are integer, and $\sum_{i \in N} b(i) = 0$. We also assume that the minimum cost flow problem has a feasible solution. The feasibility of the minimum cost flow problem can be ascertained by solving a maximum flow problem. We denote by $A(i)$ the set of arcs incident to node i, i.e., $A(i) = \{(k, l) \in A: k = i \text{ or } l = i\}$.

Basic Solutions and Optimality Conditions

The network simplex algorithm maintains a basic feasible solution at each stage. A basic solution of the minimum cost flow problem is denoted by a triple (**B**, **L**, **U**), where **B**, **L** and **U** partition the arc set A. The set **B** denotes the set of basic arcs, and **L** and **U** denote, respectively, the sets of nonbasic arcs at their lower and upper bounds. We refer to **B** as a ***basis*** and the triplet (**B**, **L**, **U**) as a ***basis structure*** of the minimum cost flow problem. It is well-known that each basis **B** of the minimum cost flow problem is a spanning tree of G. A basis structure (**B**, **L**, **U**) is called ***feasible*** if by setting $x_{ij} = 0$ for each $(i, j) \in \mathbf{L}$ and by setting $x_{ij} = u_{ij}$ for each $(i, j) \in \mathbf{U}$, the problem has a solution satisfying (2.1b) and (2.1c).

A dual solution to the minimum cost flow problem is a vector π of ***node potentials***, and a vector c^{π} of ***reduced costs*** defined as $c^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j)$ for each $(i, j) \in A$. A feasible basis structure (**B**, **L**, **U**) is called an ***optimal basis structure*** if it is possible to obtain a set of node potentials π so that the reduced

costs satisfy the following optimality conditions :

$$c^{\pi}_{ij} = 0 \text{ for each } (i, j) \in \mathbf{B}. \tag{2.2a}$$

$$c^{\pi}_{ij} \geq 0 \text{ for each } (i, j) \in \mathbf{L}. \tag{2.2b}$$

$$c^{\pi}_{ij} \leq 0 \text{ for each } (i, j) \in \mathbf{U}. \tag{2.2c}$$

Given a basis structure, the node potentials can be determined uniquely by setting $\pi(1) = 0$ and then using $(n\text{-}1)$ equations of (2.2a). A nonbasic arc (i, j) not satisfying (2.2b) or (2.2c), whichever is applicable, violates its optimality condition. We refer to an arc violating its optimality condition as an ***eligible arc***. We refer to the vector π as ***cost-potentials***.

Computing a basic solution involves only addition and subtraction of the quantities u_{ij}'s and $b(i)$'s; similarly computing the node potentials corresponding to a basis involves only addition and subtraction of costs c_{ij}'s. This is due to the fact that the underlying matrix in equations (2.1b), called the node-arc incidence matrix, is totally unimodular (i.e., each square submatrix of the matrix has determinant 0 or ±1). A detailed exposition of the property can be found, for example, in Ahuja et al. [1993] and Schrijver [1986]. As a consequence, we have the following properties.

Property 2.1 (Primal Integrality Property). If capacities of all the arcs and supplies/demands of all the nodes are integer, the minimum cost flow problem always has integral basic solution.

Property 2.2 (Dual Integrality Property). If all arc costs are integers, the node potentials corresponding to any basis of the

minimum cost flow problem are integers, and hence the reduced cost of each arc is also integer.

Example 2.1 Consider the network shown in Figure 2.1. The flow given is a basic solution.

B = {(1, 2), (3, 2), (4, 3), (5, 2), (5, 6), (7, 5)};

L = {(3, 1), (4, 5), (7, 4)};

U = {(6, 3), (1, 5)}.

Figure 2.2 gives the costs and node potentials. The reduced cost of nonbasic arcs are shown within the brackets; For basic arcs, the reduced costs are zero. Both arcs (6, 3) and (1, 5) in **U** are eligible arcs. In **L**, arc (4, 5) is eligible.

We consider the spanning tree corresponding to a basis as "hanging" from node 1. The tree arcs either are upward pointing (toward node 1) or are downward pointing (away from node 1). We associate three indices with each node i in the tree: a predecessor index, ***pred(i)***, a depth index, ***depth(i)***, and a thread index, ***thread(i)***. Each node i, other than node 1, has a unique path connecting it to node 1. The predecessor index pred(i) stores the first node in that path (other than node 1), and the depth index stores the number of arcs in the path. For node 1, these indices are zero. We say that pred(i) is the ***predecessor*** of node i, and i is the ***successor*** of node pred(i). The ***descendants*** of node i consist of node i itself, its successors, the successors of its successors, and so on. The set of descendants of any node i induces a subtree rooted at node i. We denote the nodes of the subtree by the set $D(i)$. Node i is called an ancestor of the nodes in $D(i)$. The thread indices provide a means for visiting

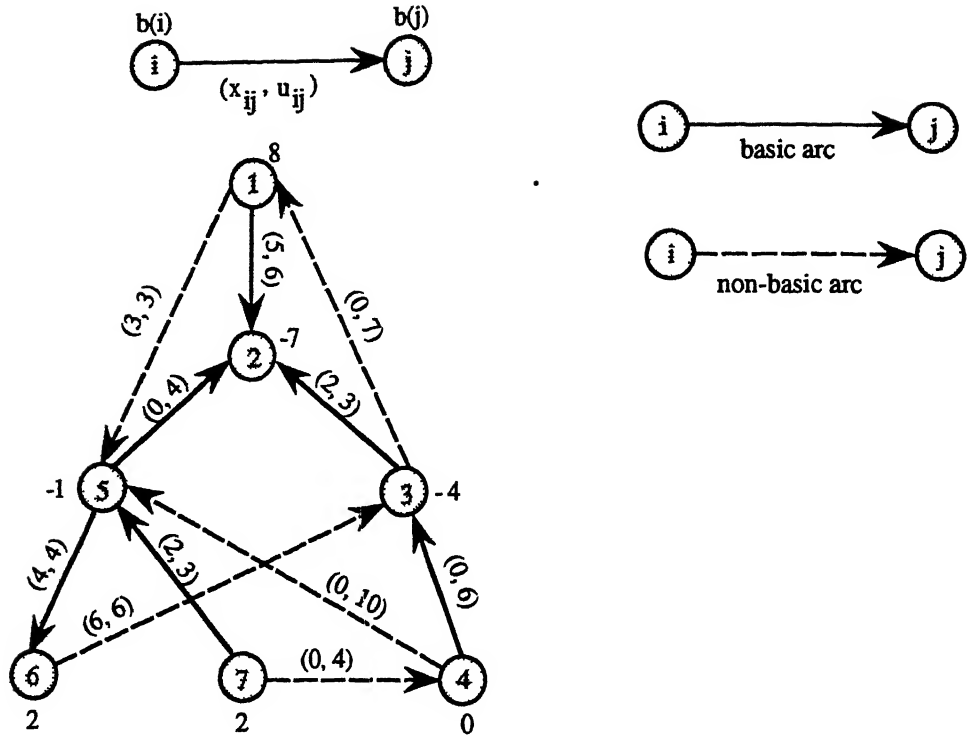


Figure 2.1 Minimum cost flow problem: a basic feasible solution

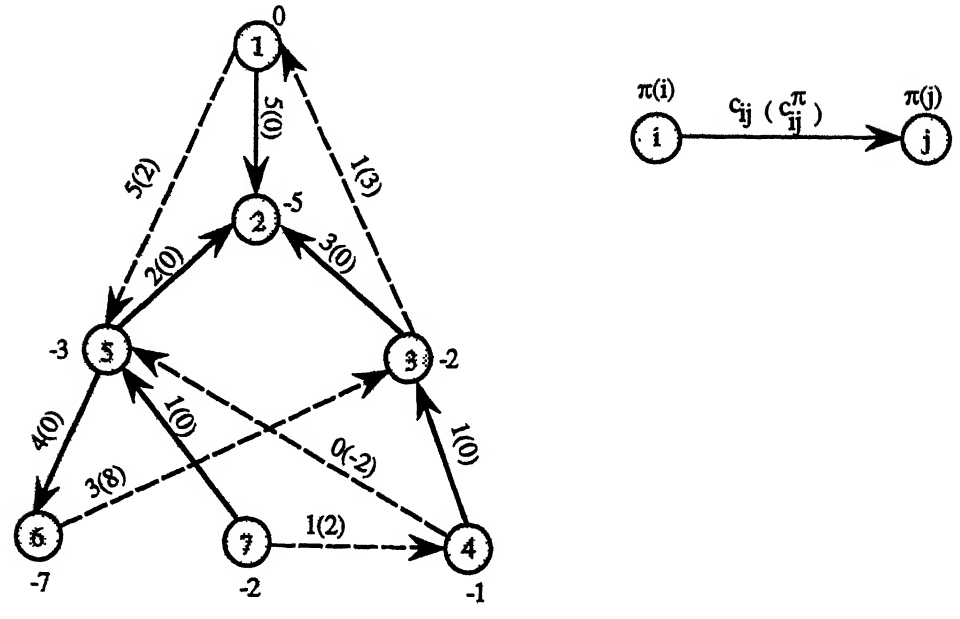
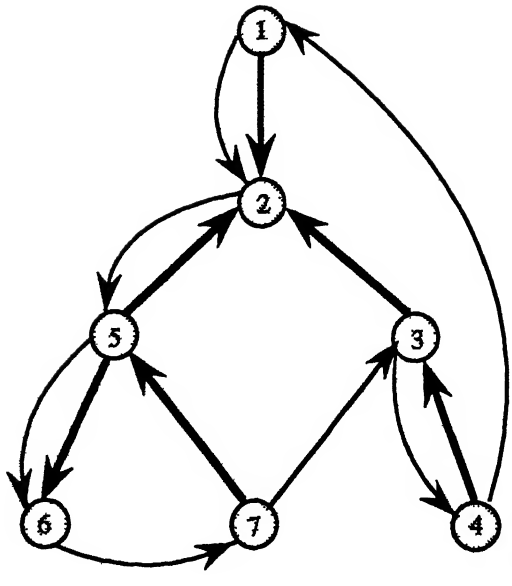


Figure 2.2 Potentials and costs for the basis in Figure 2.1

(or finding) all descendants of a node i in $O(|D(i)|)$ time. For a detailed description of the tree indices, see Kennington and Helgason [1980]. In the basis, there is a unique path connecting any two nodes. We refer to this path as the ***basis path***. The tree indices for the basis given in Figure 2.1 are given in Figure 2.3.

The network simplex algorithm works as follows. It starts with an initial basic feasible flow x and the corresponding basis structure (**B**, **L**, **U**). It selects an eligible arc (k, l) to enter the basis. Adding arc (k, l) to the basis forms a unique cycle W, we refer to this cycle as the ***pivot cycle***. If the eligible arc (k, l) is at its lower bound, then we define the orientation of the basis cycle along arc (k, l), and send maximum possible flow in the basis cycle without violating any of the lower and upper bound constraints on arcs. However, if the eligible arc (k, l) is at its upper bound, then we define the orientation of the basis cycle opposite to the direction of arc (k, l) and send maximum possible flow. To avoid this case analysis (whether the eligible arc is at its lower bound or at its upper bound) and to simplify the subsequent proofs, we shall henceforth assume that whenever x_{kl} at its upper bound enters the basis, we replace it by the variable $x'_{lk} = u_{kl} - x_{kl}$ and x'_{lk} enters the basis at its lower bound. Notice that for an eligible arc (k, l) at its upper bound, $c^{\pi}_{kl} > 0$, and when we replace it by its reversal, (l, k), its reduced cost $c^{\pi}_{lk} = - c^{\pi}_{kl}$. As a result, in our subsequent discussion we shall assume that the entering arc (k, l) is always at its lower bound and $c^{\pi}_{kl} < 0$. For example, consider the arc (6, 3) in Example 2.1, which is at its upper bound. Making the substitution $x'_{36} = u_{63} - x_{63} = 6 - x_{63}$ amounts to replacing the



i	1	2	3	4	5	6	7
pred(i)	0	1	2	3	2	5	5
depth(i)	0	1	2	3	2	3	3
thread(i)	2	5	4	1	6	7	3

Figure 2.3 Example of a tree indices:
(a) a rooted tree
(b) corresponding tree indices.

arc (6, 3) by its reversal (3, 6) with flow 0 and capacity 6. Since $c^{\pi}_{63} = 8$, $c^{\pi}_{36} = -8$.

Once the orientation of the pivot cycle W has been defined, we augment maximum possible flow, say δ, along it, given by

$$\delta = \min\{\delta_{ij} : (i, j) \in W\}, \text{ where} \qquad (2.3)$$

$$\delta_{ij} = \begin{cases} u_{ij} - x_{ij} & \text{if } (i, j) \in \overline{W} \\ x_{ij} & \text{if } (i, j) \in \underline{W}\ , \end{cases}$$

and $\overline{W}$ and $\underline{W}$ respectively denote the sets of forward arcs (i.e., those along the orientation of W) and backward arcs (those opposite to the orientation of W) in the pivot cycle. At this point some arc, on which the minimum in (2.3) is attained, in the cycle ***blocks*** further increase in the flow. The algorithm drops such a ***blocking arc*** (p, q), and obtains a new basis structure. The sequence of steps involved in moving from one basis structure to another is called a ***pivot iteration.*** A pivot iteration is ***degenerate*** if $\delta = 0$, and ***nondegenerate*** if $\delta > 0$. In Example 2.1, pivot iteration on arc (1, 5) is degenerate, i.e., $\delta = 0$, and the pivot iteration on arc (6, 3) is non-degenerate and augments $\delta = 2$ units of flow. The pivot operation might change some node potentials too. If $(k, l) \neq (p, q)$, the node potentials change. Let S and $\overline{S}$ be the subsets of nodes formed by deleting arc (p, q) from the previous basis $\mathbf{B}$, where $\overline{S}$ contains node 1. In the new basis, potentials of all nodes in $\overline{S}$ remain unchanged and potentials of all nodes in S change by a constant amount. If $k \in \overline{S}$, then the potential of each node $i \in S$ changes to $\pi(i) - c^{\pi}_{kl}$; and if $k \in S$, then the potential of each node $i \in S$ changes to $\pi(i) + c^{\pi}_{kl}$. In Example 2.1, the arc (5, 2) is the leaving arc in the

pivot iteration on arc (1, 5). Since the arc (1, 5) is at its upper bound, the arc (1, 5) is reversed; the pivot iteration is considered to be the arc (5, 1). Here $S = \{5, 6, 7\}$ and $5 \in S$. The potentials of each node i in S change to $\pi(i) + c^{\pi}_{51} = \pi(i) - 2$.

Our network simplex algorithm maintains a strongly feasible basis at every iteration. We say that a basis structure (**B**, **L**, **U**) is strongly feasible if one can send a positive amount of flow from any node in the spanning tree to node 1 along arcs in the tree without violating any of the flow bounds. An equivalent way of stating this property is that no upward pointing arc of the spanning tree can be at its upper bound and no downward pointing arc can be at its lower bound. The basis structure given in Example 2.1 is strongly feasible.

The network simplex algorithm selects the leaving arc appropriately so that the next basis is also strongly feasible. Let W be the pivot cycle formed by adding arc (k, l) to the basis tree. We call the first common ancestor of nodes k and l as the apex node of cycle W. The pivot cycle consists of the basis path from the apex node w to node k, the arc (k, l), and the basis path from node l to node w. After updating the flow, the algorithm identifies blocking arcs. If the blocking arc is unique, then it leaves the basis. If there are more than one blocking arcs, then the algorithm selects the leaving arc to be the last blocking arc encountered in traversing W along its orientation starting at node w. It can be shown (see, for example, Cunningham [1976] and Ahuja, Magnanti and Orlin [1993]) that the above rule guarantees that the next basis is strongly feasible.

We shall now observe a few facts about the degenerate pivots that will be used later. Notice that in a strongly feasible basis, we can send a positive amount of flow from any node to node 1 using the tree arcs. This implies that in a degenerate pivot, all blocking arcs will lie in the basis path from node w to node k. (Recall that we make the assumption that the entering arc (k, l) is at its lower bound). Consequently, the leaving arc will also lie in the basis path from node w to node k. As a result, the set S (as defined previously) will always contain node k and the potentials of each node $i \in S$ will change to $\pi(i) + c^{\pi}_{kl}$.

We also need to perform the degenerate pivot cleverly, else they will take too much time. We incorporate an additional test in the method described by Mulvey [1978], to identify the basis cycle for an entering arc (k, l). We trace the predecessor indices of nodes k and l backward to the root node until (i) either we locate the first common ancestor node, (ii) or we identify a blocked arc (this is the additional test). In the former case, the pivot will be nondegenerate; and in the latter case, the pivot will be degenerate. In the latter case, the time spent in the method can be charged to updating the potentials of nodes in S. In a degenerate pivot, no flow is augmented so we ignore the augmentation time. The point we wish to make is that during degenerate pivots, time to update the node potentials is the bottleneck step.

We summarize our discussion on degenerate pivots in the following properties for future reference.

Property 2.3 : In a degenerate pivot, potential $\pi(i)$ of each node $i \in S$ changes to $\pi(\dot{i}) + c^{\pi}_{kl}$, where (k, l) is the entering arc and S is the subset formed by deleting the leaving arc from the previous basis tree and not containing the root node 1.

Property 2.4 : A degenerate pivot can be performed in $O(|S|)$ time.

The following property is useful in our analysis.

Property 2.5 : Let $\frac{c}{d} \leq \frac{a}{b}$, $b > 0$ and $d > 0$. Then $\frac{a+c}{b+d} \leq \frac{a}{b}$.

Proof. The condition $\frac{a}{b} \geq \frac{c}{d}$ implies $ad \geq bc$. Adding ab to both the sides yields $a(b+d) \geq b(a+c)$, which immediately implies the property.

2.3 THE MINIMUM RATIO PIVOT RULE

The network simplex algorithm can select any eligible arc as the entering arc. Different rules for selecting the entering arc, called *pivot rules*, produce algorithms with different empirical and theoretical behaviors. In this chapter, we study the behavior of the network simplex algorithm with the minimum ratio pivot rule, which we describe next.

Let $(\mathbf{B}^{o}, \mathbf{L}^{o}, \mathbf{U}^{o})$ be the initial (strongly feasible) basis structure used in the network simplex algorithm. Let Q be the set of eligible arcs with respect to the spanning tree structure $(\mathbf{B}^{o}, \mathbf{L}^{o}, \mathbf{U}^{o})$. Recall that we have made the assumptions that all eligible arcs are at their lower bounds. Define a penalty d_{ij} for each arc $(i, j) \in A$ in the following manner :

$$d_{ij} = \begin{cases} 1, \text{ if } (i, j) \text{ is eligible.} \\ 0, \text{ if } (i, j) \text{ is not eligible.} \end{cases} \quad (2.4)$$

We point out that we use the initial spanning tree structure to define the arc penalties, which remain unchanged throughout the algorithm (similar to the arc costs).

The assumption that all eligible arcs in the initial spanning tree are at their lower bound is not restrictive. This can be overcome by defining the penalty d_{ij} of an eligible arc (i, j) at its upper bound as -1; the penalty for other arcs are defined in the same manner as in (2.4). For example, in Example 2.1, the eligible arc (4, 5) is at its lower bound and its penalty is 1. All other eligible arcs (1, 5) and (6, 3) are at their upper bound and their penalties are -1. The rest of the arcs are non-eligible and their penalties are zero. However, by replacing the arcs (1, 5) and (6, 3) by their reversals (5, 1) and (3, 6), the arcs (4, 5), (5, 1) and (3, 6) in the transformed network are eligible and at their lower bound; their penalties are 1.

Let (**B**, **L**, **U**) be an arbitrary feasible basis structure and (k, l) be an eligible arc. Adding the arc (k, l) to the basis creates a pivot cycle, which we denote by W_{kl}. Define the orientation of the cycle W_{kl} along the arc (k, l). Let C_{kl} denote the cost of the fundamental cycle W_{kl}, defined as the sum of the costs of the arcs in W_{kl} along its orientation minus the sum of the costs of the arcs in W_{kl} opposite to its orientation. Recall from Section 2.2 that the orientation of the cycle is same as that of the arc (k, l), since by our assumption eligible arc (k, l) is at its lower bound. In a similar fashion, we define the penalty of the cycle W_{kl}, denoted by D_{kl}. Then, the cost-to-penalty ratio

is denoted by $\gamma_{kl} = C_{kl}/D_{kl}$; for noneligible arcs, it is zero. The minimum ratio pivot rule always selects an eligible arc with the minimum cost-to-penalty ratio.

Let $\overline{W}_{kl}$ denote the set of arcs in W_{kl} along its orientation and $\underline{W}_{kl}$ denote the set of arcs in W_{kl} opposite to its orientation. Then, we have

$$C_{kl} = \sum_{(i,j)\in\overline{W}_{kl}} c_{ij} - \sum_{(i,j)\in\underline{W}_{kl}} c_{ij}, \tag{2.5}$$

and

$$D_{kl} = \sum_{(i,j)\in\overline{W}_{kl}} d_{ij} - \sum_{(i,j)\in\underline{W}_{kl}} d_{ij}. \tag{2.6}$$

Recall our discussion in Section 2.2, where we define the cost-potentials π as satisfying

$$\pi(1) = 0,$$

and

$$c^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j) = 0 \quad \text{for each arc } (i, j) \in \mathbf{B}. \tag{2.7}$$

If we replace each c_{ij} by c^{π}_{ij} in expression (2.5), then the expression remains intact (because $\pi(i)$'s cancel out). Next observe that each arc $(i, j) \in W$ is in $\mathbf{B}$, except the arc (k, l), and each arc $(i, j) \in \mathbf{B}$ has $c^{\pi}_{ij} = 0$. Consequently,

$$C_{kl} = c^{\pi}_{kl} \tag{2.8}$$

We also maintain a penalty-potential μ to simplify the expression D_{kl}. In a similar way as the cost-potentials π is defined, we define the penalty-potentials μ as satisfying

$$\mu(1) = 0$$

and

$$d^{\mu}_{ij} = d_{ij} - \mu(i) + \mu(j) = 0 \quad \text{for each } (i, j) \in \mathbf{B}. \tag{2.9}$$

By similar reasoning as applied to the expression for C_{kl}, we have

$$D_{kl} = d^{\mu}_{kl}. \qquad (2.10)$$

From equations (2.8) and (2.10), we have the following property.

Property 2.6 : $\gamma_{kl} = c^{\pi}_{kl}/d^{\mu}_{kl}$.

We refer to c^{π}_{kl} as the reduced cost of arc (k, l) and d^{μ}_{kl} as the reduced penalty of arc (k, l). We also point out that $\mu = 0$ at the initial iteration since $d_{ij} = 0$ for each arc in **B**. The following property is useful in our analysis.

Property 2.7 : The penalty-potentials μ (and the reduced-penalty d^{μ}) are integral, Further,

$$-(n-1) \leq \mu(i) \leq n-1 \qquad (2.11)$$

Proof. The integrality of μ (and d^{μ}) follows from the dual integrality property, since d_{ij}'s are integer. The inequality (2.11) follows from the fact $d_{ij} = 0$ or 1, and $\mu(i)$'s satisfy the equations (2.9).

Our network simplex algorithm maintains both the cost-potentials and penalty-potentials, using which the cost-to-penalty ratio of any eligible arc (k, l) can be computed in O(1) time.

2.4 BOUNDING THE NUMBER OF PIVOTS

We now analyze the worst-case complexity of the network simplex algorithm with the minimum ratio pivot rule. In this section, we first count the number of nondegenerate pivots and then count the number of degenerate pivots. In the subsequent section, we obtain the running time of the algorithm using two different data structures; array and Fibonacci heaps.

We analyze the worst-case complexity of the algorithm in terms of the parameter Δ, which is an upper bound on the sum of the arc flows in any feasible flow, i.e., $\Delta \geq \sum_{(i,j)\in A} x_{ij}$ for every feasible flow x satisfying (2.1b) and (2.1c). Our assumption that all eligible arcs are at their lower bound simplifies much of our presentation. Without loss of generality, we extend this assumption to all nonbasic arcs, i.e., all nonbasic arcs in a basic structure are at their lower bound; if necessary, we replace the arcs at their upper bound by their reversals as described in Section 2.2. Our analysis uses the following results, which we state and prove with assumption that all nonbasic arcs are at their lower bound.

Lemma 2.1 : *For the network simplex algorithm using the minimum ratio pivot rule at each iteration, each arc satisfies*

$$c_{ij}^{\pi} - \lambda d_{ij}^{\mu} \geq 0, \tag{2.12}$$

for some $\lambda < 0$.

Proof : Each basic arc (i, j) satisfies $c_{ij}^{\pi} = d_{ij}^{\mu} = 0$; hence the condition stated in the lemma holds. For nonbasic arcs, we prove this result by performing induction on the number of iterations. Consider the first iteration. Each nonbasic eligible arc (i, j) satisfies $d_{ij}^{\mu} = d_{ij} = 1$ (because $\mu = 0$). Hence (2.12) reduces to $\lambda \leq c_{ij}^{\pi}/d_{ij}^{\mu}$ for all eligible arcs. We can set λ equal to the minimum cost-to-penalty ratio among all eligible arcs in order to satisfy (2.12). We now consider any arbitrary iteration and show that if (2.12) holds before a pivot iteration, then it will hold after the pivot iteration too. In the pivot iteration, let arc (k, l) be the entering arc. Let $\alpha = c_{kl}^{\pi}$, $\beta = d_{kl}^{\mu}$, and $\lambda_o = \alpha/\beta$. Since arc (k, l) satisfies (2.12), and $c_{kl}^{\pi} < 0$, (2.12) could be true only if $d_{kl}^{\mu} > 0$. Therefore, $\lambda_o = \alpha/\beta < 0$.

We first note that (2.12) holds for $\lambda = \lambda_o$ before the pivot iteration. To see this, we use a result from parametric linear programming (see, for example, Murty [1976]) which says that if (2.12) holds for some λ for some basis, then it must hold for all $\lambda \in [\lambda_1, \lambda_2]$, $\lambda_1 \leq \lambda \leq \lambda_2$, for the same basis, where

$$\lambda_1 = \max \left\{ c^{\pi}_{ij}/d^{\mu}_{ij} \ : \ d^{\mu}_{ij} < 0 \text{ and } (i, j) \in A \right\}, \qquad (2.13)$$

and

$$\lambda_2 = \min \left\{ c^{\pi}_{ij}/d^{\mu}_{ij} \ : \ d^{\mu}_{ij} > 0 \text{ and } (i, j) \in A \right\}, \qquad (2.14)$$

Now consider the expression (2.14). Since the current basis is not optimal, there must be an eligible arc $(i, j) \in A$. For this arc (i, j), $c^{\pi}_{ij} < 0$, and since it satisfies (2.12) for some $\lambda < 0$, $d^{\mu}_{ij} > 0$. This is true for every arc currently eligible. Since $c^{\pi}_{ij}/d^{\mu}_{ij} < 0$ for this arc, we can ignore the terms corresponding to non-eligible arcs (i.e., the arcs with $c^{\pi}_{ij} \geq 0$ and $d^{\mu}_{ij} > 0$) in the expression (2.14). Consequently,

$$\lambda_2 = \min \left\{ c^{\pi}_{ij}/d^{\mu}_{ij} \ : \ (i, j) \text{ is an eligible arc} \right\},$$

That is, λ_2 is the minimum cost-to-penalty ratio, which is same as λ_o by the choice of arc (k, l). Hence (2.12) holds for $\lambda = \lambda_o$ before the pivot iteration.

Now, we prove that (2.12) holds for $\lambda = \lambda_o$ after the pivot iteration, which proves the lemma. Suppose that the pivot operation changes the potentials of the set S of nodes. It follows from our discussion in Section 2.2 that for each node $i \in S$, (i) either $\pi(i)$ and $\mu(i)$ become $\pi(i) + \alpha$ and $\mu(i) + \beta$ respectively; (ii) or $\pi(i)$ and $\mu(i)$ become $\pi(i) - \alpha$ and $\mu(i) - \beta$ respectively. We consider case (i) only; the treatment for case

(ii) is similar. Clearly, the reduced costs and reduced penalties of arcs in the cut $[S, \bar{S}]$ change, and for other arcs they remain unchanged. We consider only the forward arcs in the cut $[S, \bar{S}]$; the treatment for the backward arcs is similar. For a forward arc (i, j) in the cut, $i \in S$ and $j \in \bar{S}$, and by definition, $\lambda_o = \alpha/\beta$, or $\alpha - \lambda_o\beta = 0$, and λ_o satisfies (2.12) before the iteration. Subtracting $\alpha - \lambda_o\beta = 0$ from (2.12) yields $(c^{\pi}_{ij} - \alpha) - \lambda_o(d^{\pi}_{ij} - \beta) \geq 0$, which can be rearranged as $c^{\pi'}_{ij} - \lambda_o d^{\mu'}_{ij} \geq 0$, where π' and μ' are the modified potentials after the iteration. ■

Lemma 2.2 : *For the network simplex algorithm using the minimum ratio pivot rule, each eligible arc (i, j) satisfies $c^{\pi}_{ij} < 0$ and $d^{\mu}_{ij} > 0$.*

Proof : From Lemma 2.1, each arc (i, j) satisfies $c^{\pi}_{ij} - \lambda d^{\mu}_{ij} \geq 0$ for some $\lambda < 0$. An eligible arc (i, j) has $c^{\pi}_{ij} < 0$. Hence, if this arc has $d^{\mu}_{ij} \leq 0$, then it will violate (2.12). Consequently, $d^{\mu}_{ij} > 0$. ■

Our subsequent analysis uses counting of increases and decreases in the penalty-potentials, hence we will state a property useful in this respect. Recall Property 2.3 which states that during a degenerate pivot the cost-potential of each node $i \in S$ changes to $\pi(i) + c^{\pi}_{kl}$. Applying the same property to penalty-potential implies that during a degenerate pivot the penalty-potential of each node $i \in S$ changes to $\mu(i) + d^{\mu}_{kl}$. Since for each eligible arc (k, l), $c^{\pi}_{kl} < 0$ and $d^{\mu}_{kl} > 0$, we get the following result.

Property 2.8 : During a degenerate pivot operation, if a penalty-potential (or, cost-potential) changes, then it increases by d^{μ}_{kl} units (or decreases by $|c^{\pi}_{kl}|$ units).

An easy consequence of Property 2.8 is that the network simplex algorithm with the minimum ratio pivot rule prevents *stalling*; an exponentially long sequence of consecutive degenerate pivots (see Cunningham (1979) and Hao and Kai [1990b]). Each degenerate pivot increases some penalty-potential $\mu(i)$ by at least one unit (because d^{μ}_{kl} is an integer by Property 2.7). Since each penalty-potential satisfies $-n \leq \mu(i) \leq n$ (by Property 2.7), there can never be more than $2n^2$ consecutive degenerate pivots. Thus we have proved the following result.

Theorem 2.1 : *The network simplex algorithm with the minimum ratio pivot rule performs at most $2n^2$ consecutive degenerate pivots.*

We now obtain a bound on the number of nondegenerate pivots. Let us define the penalty of the flow as $d(x) = \sum_{(i,j)\in A} d_{ij}\, x_{ij}$ (similar to the cost of the flow, $c(x) = \sum_{(i,j)\in A} c_{ij}x_{ij}$). Since $d_{ij} = 0$ or 1 and $x_{ij} \geq 0$, we have $0 \leq d(x) \leq \sum_{(i,j)} x_{ij} \leq \Delta$. Due to the primal integrality property (Property 2.1), during a nondegenerate iteration, $\delta \geq 1$ units of flow is augmented along the pivot cycle introduced by the entering arc (k, l). By Lemma 2.2, $d^{\mu}_{kl} > 0$; since d^{μ}_{kl} are integers (Property 2.7), $d^{\mu}_{kl} \geq 1$. Therefore the penalty of the flow, $d(x)$, increases by $\delta\, d^{\mu}_{kl} \geq 1$ units in a nondegenerate pivot. The maximum possible increase in $d(x)$ is at most Δ and, consequently, the algorithm will perform at most Δ non-degenerate pivots.

To obtain a bound on the number of degenerate pivots performed by the algorithm, we need to establish the following result.

Lemma 2.3 : *Any penalty potential $\mu(i)$ increases at most $2n+\Delta$ times.*

Proof : It follows from Property 2.8 that a penalty-potential $\mu(i)$ can decrease only during a nondegenerate iteration, and if arc (k, l) is the entering arc, then it can decrease by at most d^{μ}_{kl} units. But also notice that in this iteration $d(x)$ also increases by at least d^{μ}_{kl} units. As $d(x)$ can increase by no more than Δ units, any penalty-potential $\mu(i)$ can not decrease by more than Δ units. This observation together with Property 2.7 implies that in the course of the entire algorithm, $\mu(i)$ can increase by at most $2n+\Delta$ units, thereby establishing the lemma. ■

This lemma immediately implies that the number of degenerate pivots is at most $n(2n+\Delta)$ because the network contains n nodes and each degenerate pivot increases at least one $\mu(i)$. We summarize the preceding discussion as the following theorem.

Theorem 2.2 : *The network simplex algorithm with the minimum ratio pivot rule performs at most 2Δ nondegenerate pivots and at most $n(2n+\Delta)$ degenerate pivots.*

2.5 BOUNDING THE RUNNING TIME

In this section, we obtain a bound on the worst-case running time of the algorithm. Needless to say, the running time of the algorithm depends upon the data structure used to implement the algorithm. We describe in detail the implementation of the algorithm using Fibonacci heap data structure developed by Fredman and Tarjan [1984]. Later we point out the running time of the algorithm using arrays and binary heaps.

A heap is a data structure that stores a collection of objects (which in our case are nodes), where each object has an associated index, called its ***key***. A heap is capable of performing the following operations :

find-min(i) : Find an element i with the minimum key in the heap.

delete-min(i) : Delete the element i with the minimum key in the heap.

decrease-key(i, value) : Decrease the key of element i to a smaller value, denoted by ***value***.

increase-key(i, value) : Increase the key of element i to a larger value, denoted by ***value***.

A binary heap data structure performs each of the above heap operations, except find-min (which takes $O(1)$ time), in $O(\log n)$ time, where n is the maximum number of elements present in the heap (see, for example, Cormen et al. [1990]). A Fibonacci heap data structure performs each of the find-min and decrease-key operations in $O(1)$ time, and each of the delete-min and increase-key operations in $O(\log n)$ time (see Fredman and Tarjan [1984]).

To implement our network simplex algorithm using heaps, we define the following key for each node $i \in N$:

$$\text{key}(i) = \min\{\gamma_{ij} : (i, j) \in Q\} \qquad (2.15)$$

where $\gamma_{ij} = c^{\pi}_{ij}/d^{\mu}_{ij}$ and Q is the set of outgoing eligible arcs at node i. In other words, the key of node i is the minimum ratio among outgoing eligible arcs at node i. If node i has no eligible arc emanating from it, then we set key(i) = 0. We also maintain another index, ***arc(i)***, for each node $i \in N$; arc(i) stores the arc

emanating from node i whose cost-to-penalty ratio equals key(i). If key(i) = 0, then arc(i) is undefined. In this implementation, the algorithm will work in the following manner. We perform a find-min operation to identify a node, say node k, with the minimum key; the arc(k) gives the arc having the minimum key. If the minimum key is zero, then there is no eligible arc and we terminate the algorithm. Suppose that the minimum key is nonzero. Let (k, l) = arc(k). Next we perform a pivot operation with arc (k, l) as the entering arc. Suppose that the pivot operation changes the potential of a subset S of nodes. This changes the cost-to-penalty ratios of arcs in the cut [S, $\bar{S}$]. As a result, keys of several nodes need to be updated. We will study the effort needed to make these changes separately for degenerate and nondegenerate pivots.

For a nondegenerate pivot, find-min operation requires O(1) time. But augmenting the flow along the basis cycle and updating the node potentials and tree indices takes $O(n)$ time. We also compute the key of each node afresh, which requires a total of $O(m)$ time. The modified key of a node i may be lower than (case 1), or higher than (case 2), or equal to (case 3) its previous key. To update the Fibonacci heap, in case 1 we perform a decrease-key operation, in case 2 we perform an increase-key operation and in case 3 we do nothing. In the worst-case, we perform n increase-key operations requiring a total of O(n log n) time. Considering the times of all the operations, we can conclude that we can execute a nondegenerate pivot iteration in O(m + n log n) time. As the algorithm performs at most Δ nondegenerate pivot iterations, the total time spent on

nondegenerate pivots is $O(\Delta(m + n \log n))$.

We next consider the time spent on degenerate pivots. As earlier, find-min operation takes $O(1)$ times. We now use Property 2.4 which states that the time to perform a degenerate pivot can be charged to updating the potentials of nodes in S. In degenerate pivots, we do not recompute the key of every node in N (because it will take too much time), but recompute the keys of only those nodes whose potentials change during the pivot, namely, the nodes in S. Recomputing the keys of nodes in S takes $O(\sum_{i \in S} |A(i)|)$ time, and updating the Fibonacci heap takes $O(|S| \log n)$ time. Updating the potentials of nodes in S modifies the cost-to-penalty ratios of arcs in $[S, \bar{S}]$ due to which the keys of some nodes in $\bar{S}$ will also change. We shall now prove a crucial observation that the keys of nodes in $\bar{S}$ either stay the same or they decrease. This observation will allow us to update the keys of nodes in $\bar{S}$ by performing decrease-key operations which are much more efficient than increase-key operations.

Consider any node $i \in \bar{S}$. As earlier, let Q be the set of eligible arcs. If for each arc (i, j) emanating from node i, $j \in \bar{S}$, then key(i) remains unchanged; otherwise it might change. Arc (i, j) will satisfy one of the three following conditions : (i) it is ineligible before the pivot and it remains ineligible; (ii) it is ineligible before the pivot but it becomes eligible after the pivot; and (iii) it is eligible before and also after the pivot. In the first case, the cost-to-penalty ratio of arc (i, j) remains zero and in the second case it strictly decreases. We now consider the third case. Suppose that there is an arc (i, j) emanating from node i for which $j \in S$. It follows from Property

2.8 that the potentials $\pi(j)$ and $\mu(j)$ become $\pi(j) + c^{\pi}_{kl}$ and $\mu(j) + d^{\mu}_{kl}$ after the pivot operation. These changes will modify the reduced cost of arc (i, j) to $c^{\pi}_{ij} + c^{\pi}_{kl}$ and the reduced penalty of arc (i, j) to $d^{\mu}_{ij} + d^{\mu}_{kl}$. Hence the modified cost-to-penalty ratio of arc (i, j) will be $(c^{\pi}_{ij} + c^{\pi}_{kl})/(d^{\mu}_{ij} + d^{\mu}_{kl})$. Using the facts (i) $c^{\pi}_{ij}/d^{\mu}_{ij} \geq c^{\pi}_{kl}/d^{\mu}_{kl}$, (ii) $d^{\mu}_{ij} > 0$, and (iii) $d^{\mu}_{kl} > 0$, we obtain $(c^{\pi}_{ij} + c^{\pi}_{kl})/(d^{\mu}_{ij} + d^{\mu}_{kl}) \leq c^{\pi}_{ij}/d^{\mu}_{ij}$. [Here we make use of the Property 2.5 that if $\frac{c}{d} \leq \frac{a}{b}$, $b > 0$, $d > 0$, then $\frac{a+c}{b+d} \leq \frac{a}{b}$]. The preceding discussion establishes that in a degenerate pivot, keys of nodes in $\bar{S}$ either stay the same or decrease. We can update the keys for node in $\bar{S}$ in the following manner. We consider each node $j \in S$ one by one, and for each such node under consideration we examine incoming arcs at node j one by one. If for any arc (i, j), $i \in \bar{S}$, we recompute the cost-to-penalty ratio of the arc. If this ratio is less than key(i), we perform a decrease-key operation requiring $O(1)$ time. We also update arc(i). Clearly, using this method, we can update the keys of nodes in $\bar{S}$ in a total of $O(\sum_{i \in S}|A(i)|)$ time.

We are now in a position to bound the time taken by the algorithm during the degenerate iterations. We have shown thus far that if during a degenerate iteration, potentials of a subset S of nodes change, then updating the keys and the Fibonacci heap requires $O(|S|\log n + \sum_{i \in S}|A(i)|)$ time. Now observe that in all the degenerate iterations, any node can belong to S at most $O(n+\Delta)$ times because each time a node belongs to S its potential changes and, by Lemma 2.3, the potential of any node can change at most $2n+\Delta$ times. This implies that (i) if we sum $|S|$ over all the iterations, then it would be no more than $n(2n+\Delta)$ and (ii) if we

sum $\sum_{i \in S} |A(i)|$ over all the iterations then it would be no more than $m(2n+\Delta)$. Under the reasonable assumption that $\Delta \geq n$, these observations give a bound of $O(\Delta(m+n \log n))$ on the time taken by degenerate iterations, which is the same as the time taken by the nondegenerate iterations. We can summarize our discussion thus far in the form of the following theorem.

Theorem 2.3 : *The network simplex algorithm using the minimum ratio pivot rule performs $O(n\Delta)$ pivots and can be implemented in $O(\Delta(m+n \log n))$ time.*

Binary heap implementation

To implement the algorithm in a binary heap data structure, we can create a binary heap with arcs in A as data item; each arc (i, j) in the heap has its cost-to-penalty ratio γ_{ij} as its key. In each iteration, we select an arc with the minimum cost-to-penalty ratio by selecting the arc with min-key in the binary heap. As we pointed out at the beginning of this section, all heap operations find-min, delete-min, increase-key, and decrease-key, take $O(\log m) = O(\log n)$ time in a binary heap.

In each non-degenerate iteration, we compute the key of all arcs afresh, which takes $O(m \log n)$ time; the time required for all other operations updating node potentials and augmenting the flow, is dominated by this time. Since the algorithm performs at most Δ nondegenerate pivots (Theorem 2.2), the time required for nondegenerate iterations is $O(\Delta\, m \log n)$.

We update the keys of the arcs in degenerate iterations by scanning the adjacency list $A(i)$ of each node i in the set S (the set of nodes for which node potential changes). Note that the key

of only arcs in the cut $[S, \bar{S}]$ change. When a node i is in S, its $\mu(i)$ increases, which happens at most $(2n+\Delta)$ time by Lemma 2.3. Thus the number of times all arcs in A are scanned in degenerate pivots is the sum of the expression $\sum_{i \in S} |A(i)|$ over all degenerate iterations, which is $O(m\Delta)$. All operations in a degenerate iteration, except updating keys, takes $O(|S|$ time, hence this time can be charged to arc scanning time, which is $O(\Delta\ m)$. Since updating a key of an arc once takes $O(\log\ n)$ time and all arc change their keys at most $O(\Delta\ m)$ times, the time for updating keys is $O(\Delta\ m \log\ n)$. Consequently, the time taken for degenerate pivots is $O(\Delta\ m \log\ n)$, which is the same as that for nondegenerate pivots. Therefore, the algorithm runs in $O(\Delta\ m \log\ n)$ time, when implemented using binary heaps.

Array implementation

The array implementation of the algorithm is same as the Fibonacci heap implementation, except that the key for each node in N is stored in an array of n elements. In the array data structure, finding the minimum key takes $O(n)$ time; the operations decrease-key and increase-key operations take $O(1)$ time. Except for the time taken for the min-key operations in degenerate pivots, the time bound for other operations remains intact. Since the number of degenerate pivots is $O(n\ \Delta)$ (Theorem 2.2), the time taken for min-key operations in degenerate pivots is $O(n^2\Delta)$. Consequently, the algorithm runs in $O(n^2\Delta)$ time.

The time bound we obtain for the network simplex algorithm with the minimum ratio pivot rule is pseudopolynomial. The algorithm, however, would run in polynomial time if Δ is

polynomial in terms of the input size parameters. As discussed in the next section, the assignment and shortest path problems have $\Delta = n$, and the algorithm solves those problems in polynomial time.

2.6 SPECIFIC IMPLEMENTATION

In this section, we specialize the network simplex algorithm and its complexity results to the assignment and shortest path problems.

Assignment Problem

The assignment problem is a minimum cost flow problem (2.1) on a bipartite network $G = (N_1 \cup N_2, A)$ such that $|N_1| = |N_2| = n/2$, $A \subseteq N_1 \times N_2$, $b(i) = 1$ for all $i \in N_1$ and $b(i) = -1$ for all $i \in N_2$. The special structure of the assignment problem implies that in any basic feasible solution, $n/2$ arcs carry unit flow and the remaining arcs have zero flow (see, for example, Barr, Glover and Klingman [1977]). Therefore, for the assignment problem, $\Delta = n/2$. Specializing the results of the preceding sections yields that when applied to the assignment problem, the network simplex algorithm with the minimum ratio pivot rule will perform $O(n)$ nondegenerate pivots, $O(n^2)$ degenerate pivots and would run in $O(nm + n^2 \log n)$ time.

Shortest Path Problem

We next study the performance of our network simplex algorithm on the shortest path problem. The shortest path problem is to identify a path of minimum cost from a specified *source* node s to another specified *sink* node t. The shortest path problem can

be formulated as a minimum cost flow problem (2.1) by setting $b(s)$ = 1, $b(t)$ = -1, and $b(i)$ = 0 for the remaining nodes. A feasible solution of the shortest path problem consists of unit flow along a directed path from node s to node t and zero flow on rest of the arcs. Therefore, similar to the assignment problem, the shortest path problem satisfies the property that $O(n)$ arcs carry unit flow and the remaining arcs have zero flow. Consequently, our network simplex algorithm will solve the shortest path problem within $O(n^2)$ pivots and in $O(nm + n^2 \log n)$ time. We summarize our discussion with the following result :

Theorem 2.4 : *The network simplex algorithm using the minimum ratio pivot rule solves the assignment problem as well as the shortest path problem within $O(n^2)$ pivots and in $O(nm + n^2 \log n)$ time.*

We point out that in the network simplex algorithm for the shortest path problem every basis is a directed out-tree rooted at the source and the optimal basis is a shortest path tree (see, for example, Ahuja, Magnanti and Orlin [1993]). In the shortest path tree, the unique path from node s to every other node is a shortest path. Hence, the network simplex algorithm finds a shortest path from node s to every other node of the network.

Chapter 3

CYCLE-CANCELLING ALGORITHM FOR THE MINIMUM COST FLOW PROBLEMS

3.1 INTRODUCTION

One of the most straightforward method for solving the minimum cost flow problem is the cycle cancelling algorithm. The algorithm repeatedly finds a negative augmenting cycle (equivalently, a directed cycle in the residual network $G(x)$) and cancels this cycle by augmenting maximum possible flow along the cycle.

The cycle-cancelling algorithm is credited to Klein [1967]. The first polynomial as well as strongly polynomial cycle-cancelling algorithm is the minimum mean cycle-cancelling algorithms due to Goldberg and Tarjan [1988]. Their algorithm repeatedly finds the minimum mean cycle in the residual network and cancels the cycle until an optimum solution is obtained, the algorithm runs in $O(n^2m^2 \min\{\log(nC), m \log n\})$. A variation of this algorithm, which they call as the algorithm cancel-and-tighten, runs in $O(\{m \log n \min\{\log n C, mn \log n\})$. Barahona and Tardos [1989] developed an algorithm, which repeatedly finds and cancels a collection of node-disjoint cycles. Their algorithm requires solving an assignment problem at each iteration, and runs in $O(m^2 \log (mCU)(m+n \log n))$ time. The third

type of cycle-cancelling algorithm is due to Wallacher and Zimmerman [1991], which augment flows along minimum ratio cycles and runs in $O(nm^2 \log (nCU))$ time. Wallacher [1991] has generalized the idea of cancelling minimum ratio cycles and proposed different variants of this algorithm. Currently, the best available time bound for the minimum cost flow problem is $O(\min\{nm \log(n^2/m) \log(nC), nm(\log \log U) \log (nC), m \log n (m+n \log n)\})$; the three bounds in this expression are, respectively, due to Goldberg and Tarjan [1987], Ahuja, Goldberg, Orlin, and Tarjan [1992], and Orlin [1988]. Even though none of the above cycle-cancelling algorithm match this time bound, their interest lie in the fact that they are simple and yet run in polynomial time.

In this chapter, we propose two variants of cycle-cancelling algorithms. Our algorithms work with a feasible flow x and keep node potential $\pi(i)$ for every node i. We work with the residual network $G(x)$ which has an arc (i, j) with cost $c^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j)$ and residual capacity $u_{ij} - x_{ij}$ for every arc (i, j) with $x_{ij} < u_{ij}$, and an arc (j, i) with cost $-c^{\pi}_{ij}$ and residual capacity for every arc (i, j) with $x_{ij} > 0$. We call arcs with negative reduced costs in the residual network as ***eligible arcs***, and the subgraph of the residual network consisting of eligible arcs as the ***admissible network***. We say a node is ***reachable*** from node p if it has a directed path from node p.

Our first cycle-cancelling algorithm uses capacity scaling. It can be considered as the primal version of capacity scaling algorithms due to Edmonds and Karp [1972] and Orlin [1988]. The algorithm chooses an eligible arc with maximum residual capacity,

say arc (q, p). We then find a shortest path from node p to node q in the residual network using only noneligible arcs with "sufficiently large" residual capacities; if the cycle consisting of the path and the arc (q, p) is negative, we augment flow along this cycle. We repeat this process until there is no eligible arc. The algorithm solves the minimum cost flow problem with integral supply/demands and capacities in $O(m \log U (m + n \log n))$ time, the same as that of the above mentioned algorithms.

Our second cycle-cancelling algorithm uses a cancel subroutine of Goldberg and Tarjan's cancel-and-tighten algorithm [1988] and uses ideas such as ε-optimality and fixing arc flows. The algorithm runs in $O(nm \log n \min\{\log(nC), m \log n\})$ time. Each iteration starts with a acyclic admissible graph. We then sufficiently increase potentials of nodes reachable through eligible arcs from a node that has a most negative (eligible) incoming arc. Then the algorithm makes the new admissible graph acyclic using the cancel subroutine of Goldberg and Tarjan's algorithm. To make the algorithm strongly polynomial, it is sufficient to update the potentials at the start of an iteration by applying a shortest path algorithm for nonnegative costs. Our algorithm differs from Goldberg and Tarjan's cancel-and-tighten algorithm in the following two aspects :

(1) Updating potentials in each iterations is based on a different intuitive idea. Our algorithm finds a cut whose forward arcs are noneligible. Then the optimality violation of the backward arcs in the cut is shared with the forward arcs, improving the maximum optimality violation over all arcs in the cut by a factor of two; Goldberg and Tarjan's

algorithm improves the maximum optimality violation over all arcs in the entire graph (ε-optimality) by a factor of n.

(2) To make the algorithm strongly polynomial, they solve a minimum mean cycle problem; we solve a shortest path problem with nonnegative costs.

The organization of this chapter is as follows :

In Section 3.2, we review some relevant definitions and results. In Section 3.3, we propose the cycle-cancelling algorithm with capacity scaling and present the analysis of the algorithm. In Section 3.4, we propose the algorithm cancel-and-share. We present both polynomial and strongly polynomial implementations of this algorithm along with their complexity analysis.

3.2 THE BACKGROUND

Our framework for discussion in this chapter is as follows. As usual, let $G = (N, A)$ be a directed network with node set N containing n nodes and arc set A containing m arcs. We associate a cost c_{ij} and a capacity u_{ij} with every arc $(i, j) \in A$. We associate with each node $i \in N$ a number $b(i)$ which indicates its supply or demand depending on whether $b(i) > 0$ or $b(i) < 0$. We assume without loss of generality that the minimum cost flow problem defined on this network has a feasible solution. We denote the largest magnitude of any arc cost by C, and the largest magnitude of any finite arc capacity by U.

The minimum cost flow problem has unbounded solution if and only if G has a negative (directed) cycle consisting only of

uncapacitated arcs. Since this can be detected by applying a shortest path algorithm, without loss of generality, we can assume G has no negative cycle consisting of uncapacitated arcs. We assume that, whenever arc $(i, j) \in A$, arc $(j, i) \notin A$.

The residual network

Our cycle-cancelling algorithms work on the residual network. For a given flow x, we construct the residual network $G(x)$ as follows. For each arc $(i, j) \in A$ with arc flow $x_{ij} < u_{ij}$, we introduce an arc (i, j) in $G(x)$ with residual capacity $u_{ij} - x_{ij}$ and cost c_{ij}; for each arc $(i, j) \in A$ with arc flow $x_{ij} > 0$, we introduce an arc (j, i) in $G(x)$ with residual capacity x_{ij} and cost $c_{ji} = -c_{ij}$. We denote the arc set of $G(x)$ by $A(x)$. Note that $A(x) = A^+ \cup A^-$, where

$$A^+ = \{(i, j) : (i, j) \in A, x_{ij} < u_{ij}\}$$
$$A^- = \{(i, j) : (j, i) \in A, x_{ji} > 0\}.$$

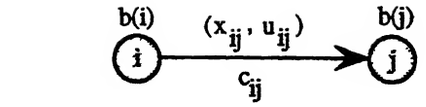
Each augmenting cycle in G with respect to a flow x corresponds to a directed cycle in $G(x)$, and vice versa (here we choose to ignore the cycles of the type $(i, j) \cup (j, i)$ in $G(x)$). The cost of a directed cycle W in $G(x)$ is defined as

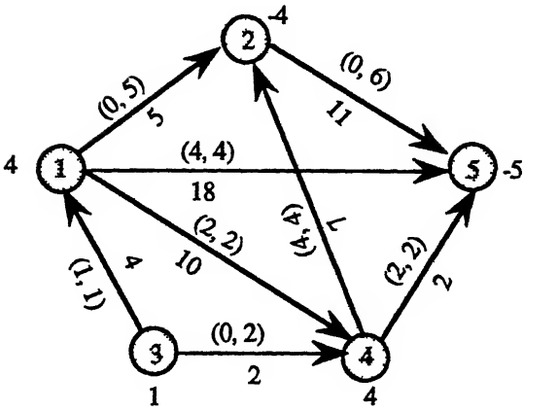
$$c(W) = \sum_{(i,j) \in W} c_{ij}, \qquad (3.1)$$

and it is the cost of the corresponding augmenting cycle in G. Figure 3.1(b) gives the residual network for the flow on example network in Figure 3.1(a).

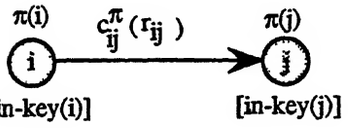
Optimality Conditions

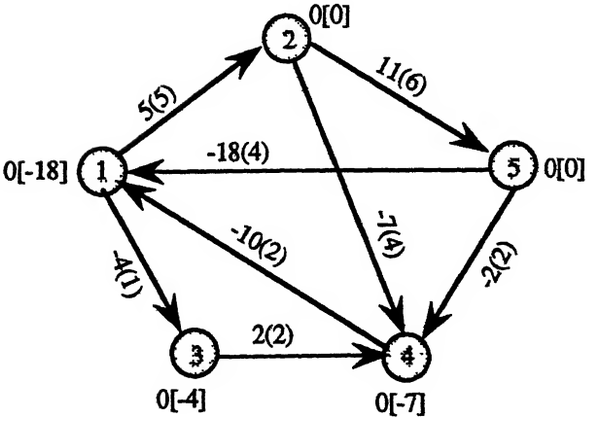
The following well-known optimality conditions are fundamental in the development of cycle-cancelling algorithms. The





(a)





(b)

Figure 3.1 Minimum cost flow problem:
(a) Example network.
(b) The residual network for flow in Figure 3.1(a)

proofs of these conditions can be found in Ahuja et al. [1993].

Theorem 3.1 (Negative Cycle Optimality Conditions). *A feasible solution x is an optimal solution of the minimum cost flow problem if and only if the residual network $G(x)$ contains no negative cost (directed) cycle.*

Theorem 3.2 (Reduced Cost Optimality Conditions). *A feasible solution x is an optimal solution of the minimum cost flow problem if and only if some potential π satisfies the following reduced cost optimality conditions:*

$$c_{ij}^{\pi} \geq 0 \text{ for every arc } (i, j) \text{ in } G(x), \tag{3.2}$$

where $c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j)$.

Approximate Optimality

A flow x is said to be ε-*optimal* for some $\varepsilon > 0$ if x, together with some node potential vector π, satisfies the following ε-*optimality conditions*:

$$c_{ij}^{\pi} \geq -\varepsilon \quad \text{for every arc } (i, j) \text{ in } G(x). \tag{3.3}$$

The concept of ε-optimality is due to Bertsekas [1979] and Tardos [1985].

We note that, for a flow x together with some node potential vector π, if $-\varepsilon$ denotes the value of the reduced cost of the most negative arc, then the flow x is ε-optimal. For example, the flow on the example network given in Figure 3.1 is ε-optimal, for $\varepsilon = 18$ and $\pi = 0$.

all nodes which are not reachable from node p, we let $d(i) = M$, where M is an arbitrarily large positive real number greater than or equal to $|c^{\pi}_{qp}|$. This means $d(i)$'s are shortest distances from node p in the augmented graph obtained from $G(x, r_{qp}/2, \pi)$ by adding arcs (p, i) for each node i with cost M. These $d(i)$'s satisfy $c^{\pi}_{k\ell} + d(k) - d(\ell) \geq 0$ for every arc (k, ℓ) in the augmented graph, and hence also for every arc (k, ℓ) in $G(x, r_{qp}/2, \pi)$. Therefore, without loss of generality, we refer to these $d(i)$'s as shortest distances from node p, and we assume they satisfy the shortest path optimality conditions. Let P_{pq} denotes a shortest path from p to q in $G(x, r_{pq}/2, \pi)$ found in the above process. We update the potential π to $\hat{\pi} = \pi - d$. If $c^{\pi}_{qp} \geq 0$, we do nothing; otherwise, we cancel the cycle $W := (q, p) \cup P_{pq}$ by augmenting the maximum possible flow. Figure 3.2 presents an algorithmic description of the cycle-cancelling algorithm with capacity scaling.

The justification for the algorithm is straightforward. Whenever q is not reachable from P in $G(x, \Delta, n)$ in an iteration, we have $c^{\pi}_{qp} \geq 0$ after updating π to $\pi - d$ (since $d(q) = M \geq |c^{\pi}_{qp}|$ and $d(p) = 0$). Therefore, we would be able to find a shortest path from node p to node q if $c^{\pi}_{qp} < 0$. Note that the cycle W has the cost c^{π}_{qp} after the update. The algorithm terminates when $c^{\pi}_{ij} \geq 0$ for every arc (i, j) in $G(x)$, which implies (by reduced cost optimality conditions) that the flow x is an optimal flow.

Analysis of the algorithm

Each iteration of the algorithm selects an eligible arc with the maximum residual capacity; we call this arc as the arc

algorithm *cycle cancelling with capacity scaling;*

begin

π: = 0

let x be a feasible integral flow;

while there is an arc (i, j) with $c_{ij}^{\pi} < 0$ **do**

begin

find an eligible arc (q, p) (i.e., $c_{qp}^{\pi} < 0$)

with the maximum residual capacity;

Δ: = $\frac{r_{qp}}{2}$;

find the shortest path distances $d(i)$ to every

node i from node p in $G(x, \Delta, \pi)$;

π: = π-d;

if $c_{qp}^{\pi} < 0$ **then**

begin

let P_{pq} denote a shortest path from node p to

node q in $G(x, \Delta, \pi)$;

let $\delta = \min\{r_{ij}: (i, j) \in (q, p) \cup P_{pq}\}$;

augment δ units of flow along the negative cycle

W: = $(q,p) \cup P_{pq}$;

update the flow x and $G(x)$;

end;

end;

end;

Figure 3.2. Cycle cancelling algorithm with capacity scaling.

examined. We bound the number of iterations. Consider an iteration in which the arc (q, p) is examined. Let $G^1 = G(x, r_{qp}/2, \bar{\pi})$, where $\bar{\pi}$ is the potential and r_{qp} is the residual capacity of the arc (q, p) at the start of the iteration. Let $d(.)$ denote the shortest path distance from p to other nodes with respect to the arc cost c^{π}_{ij}. By shortest path optimality conditions, (i) $c^{\bar{\pi}}_{ij} + d(i) - d(j) \geq 0$ for all arcs in G^1, and (ii) $c^{\bar{\pi}}_{ij} + d(i) - d(j) = 0$ for all arcs in the shortest path P_{pq} from node p to node q. The new potential at the end of the iteration is $\hat{\pi} = \pi - d$; thus, the statements (i) and (ii) imply : (iii) $c^{\hat{\pi}}_{ij} \geq 0$ for all arcs in G^1, and (iv) $c^{\hat{\pi}}_{ij} = 0$ for all arcs in the path P_{pq}.

By definition, G^1 contains all arcs in $G(x)$ with $c^{\bar{\pi}}_{ij} \geq 0$ and $r_{ij} \geq r_{qp}/2$. The choice of the arc (q, p) implies that $c^{\bar{\pi}}_{ij} \geq 0$ for all arcs in $G(x)$ with $r_{ij} > r_{qp}$ and these arcs are in G^1; so, by (iii) $c^{\hat{\pi}}_{ij} \geq 0$ for these arcs. When $c^{\hat{\pi}}_{qp} < 0$, we augment flow on the cycle $W := \{(q, p)\} \cup P_{pq}$. As a result, the residual capacity of the reversals of the arcs in W increases. By the fact that $c^{\hat{\pi}}_{pq} > 0$ and the statement (iv), it follows that (v) $c^{\hat{\pi}}_{ij} \geq 0$ for the arcs in $G(x)$ whose residual capacity increases in the iteration. Consequently, $c^{\hat{\pi}}_{ij} \geq 0$ for every arcs (i, j) in $G(x)$ with $r_{ij} > r_{qp}$ at the end of the iteration. Therefore, the residual capacity of the arc examined in the next iteration is at the most r_{qp}. Repeating this argument inductively in the subsequent iterations, we conclude the following lemma.

Lemma 3.2 : ***The maximum residual capacity of the arcs violating optimality conditions at each iteration is nonincreasing.***

Now consider any arc (i, j) in G^1. By definition of G^1, $c_{ij}^{\overline{\pi}} \geq 0$ and $r_{ij} \geq r_{qp}/2$ at the beginning of the iteration. By the statement (iii), $c_{ij}^{\pi} \geq 0$ at the end of the iteration. Extending the argument inductively for subsequent iterations, we conclude that (vi) the arc (i, j) may become eligible (i.e., $c_{ij}^{\pi} < 0$) at the end of a subsequent iteration only if r_{ij} less than $r_{kl}/2$, where (k, l) is the arc being examined at this iteration. By Lemma 3.2, $r_{kl}/2 \leq r_{qp}/2$.

We have either $\hat{c}_{qp}^{\pi} \geq 0$ or $\hat{c}_{qp}^{\pi} < 0$ at the end of the iteration at which the arc (q, p) is examined. In the later case the new residual capacity of the arc (q, p) is less than $r_{qp}/2$, since we augment at least $\lceil r_{qp}/2 \rceil$ units of flow along the cycle W. Note that units of flow augmented is integral. In the former case, when we have $c_{qp}^{\pi} < 0$ at a subsequent iteration, by the statement (vi) and the following observation, the new residual capacity of arc (q, p) at that iteration is less than or equal to $\lceil r_{qp}/2 \rceil$. Thus, in both cases the residual capacity of the arc (q, p) is less than or equal to $r_{qp}/2$ at the first subsequent iteration at which $c_{qp}^{\pi} < 0$. If the residual capacity of the arc (q, p) increases at a latter iteration by the statement (iv), $c_{qp}^{\pi} \geq 0$. By repeating the argument, we conclude that if $c_{qp}^{\pi} < 0$ at any subsequent iteration, the residual capacity of the arc (q, p) at that iteration is less than $r_{qp}/2$. The arc (q, p) is again examined only when $c_{pq}^{\pi} < 0$. Hence we have the following lemma.

Lemma 3.3 : ***When an arc in G(x) is reexamined at an iteration, its residual capacity is less than or equal to half of its value when it was examined previously.***

Theorem 3.3 : *The algorithm solves the minimum cost flow problem with integral are capacities in O(m log U) iterations, where U is the largest capacity of arcs.*

Proof : Since the starting flow is integral, the flow augmented at any iteration is integral; consequently, the residual capacities of the arcs in $G(x)$ are integral. The residual capacity of an arc (k, l) when it is examined for the first time is less than or equal to u_{kl}, since r_{kl} is either x_{kl} or $u_{kl} - x_{kl}$ and $0 \le x_{kl} \le u_{kl}$. The arc (k, l) is examined at most $\log u_{kl} = O(\log U)$ times, since by Lemma 3.3, the (integral) residual capacity reduces to half its former value whenever it is reexamined. Since we consider at most 2m arcs in G(x) throughout the algorithm, the number of iterations is bounded by $O(m \log U)$ time. ■

Each iteration of the algorithm selects an eligible arc (q,p) with the maximum residual capacity, which requires $O(m)$ time. Further, each iteration includes (a) solving a shortest path problem for node p to every node in N in the network $G(x, r_{qp}/2, \pi)$, (b) updating the potential π, and (c) augmenting the flow along the cycle, $W = (q,p) \cup P_{pq}$, if $c^{\pi}_{qp} < 0$ after updating the potential. By definition, $c^{\pi}_{ij} \ge 0$ for every arc in $G(x, r_{qp}/2, \pi)$. Thus, the operation (a) can be done by applying Dijkstra's algorithm for finding shortest path with nonnegative arc lengths. The implementation of the Dijkstra's algorithm due to Fredman and Tarjan [1984], runs in $O(m + n \log n)$ time. Operations (b) and (c) require $O(n)$ time. Thus operation (a) dominates running time of all other operations in an iteration; therefore, each iteration

runs in $O(m + n \log n)$ time. We have the following corollary by Theorem 3.3.

Corollary 3.1 : *The proposed algorithm runs in $O((m \log U)(m+n \log n)$ time.*

3.4 A CYCLE-CANCELLING ALGORITHM BY SHARING OPTIMALITY VIOLATION

For a feasible flow x and potential π, we define the admissible graph as the subgraph of the residual network $G(x)$ consisting of eligible arcs (i.e., arcs with $c_{ij}^{\pi} < 0$). We call the (directed) cycles in the admissible graph as admissible cycles. Our algorithm uses the idea of cycle-cancelling algorithm, the algorithm cancel-and-tighten, due to Goldberg and Tarjan [1988]. We start with a feasible flow x and an arbitrary potential π, which is by default 0. We cancel admissible cycles until the admissible graph becomes acyclic. We then try to create new eligible arcs in $G(x)$ so that the admissible graph becomes cyclic; we do this by updating node potentials in a way backward arcs in a cut share their optimality violation with forward arcs in the cut. This process is repeated until an optimum solution is obtained. Our algorithm reduces ε-optimality by a factor of at least 2 over a cut, whereas Goldberg and Tarjan's algorithm reduces ε-optimality by a factor of at least n over the entire residual network. We obtain a strongly polynomial implementation by solving a shortest path problem with nonnegative costs, where they obtain a strongly polynomial implementation by solving a minimum mean cycle problem after every n iterations. Since both algorithms reduce ε-optimality by a factor of at least two, in at most n consecutive iteration, both algorithms have the same time complexity.

We first find a feasible flow, and initialize the potential π to 0; we also make the initial admissible graph acyclic. We then choose a most negative (eligible) arc (q, p) in $G(x)$ and find the set of nodes S reachable from node p through eligible arcs. Let $\bar{S} = N\text{-}S$; $\bar{S} \neq \phi$ since the admissible graph is acyclic. Thus $[S, \bar{S}]$ is a cut in $G(x)$. Let $(S, \bar{S})$ denotes the set of forward arcs in the cut $[S, \bar{S}]$ and $(\bar{S}, S)$ denote the set of backward arcs in the cut $[S, \bar{S}]$. Note that $p \in S$ and $q \in \bar{S}$. Let $\varepsilon = -c^{\pi}_{qp}$. We update $\pi(i)$ to $\pi(i) + \varepsilon/2$ for every node $i \in S$. As a consequence, (i) the reduced cost of every arc in $(S, \bar{S})$ reduces by $\varepsilon/2$, and (ii) the reduced cost of every arc in $(\bar{S}, S)$ increases by $\varepsilon/2$. Note that, by construction of the cut $[S, \bar{S}]$, the arcs in $(S, \bar{S})$ are noneligible (i.e., $c^{\pi}_{ij} \geq 0$) before updating the potential π. Updating potential π may create new eligible arcs in $(S, \bar{S})$ while keeping at least the arc (q, p) in $(\bar{S}, S)$ eligible. Thus, by updating the potential in the above manner, we try to create new eligible arcs by sharing the optimality violations of eligible arcs in $(\bar{S}, S)$ with (noneligible) arcs in $(S, \bar{S})$ in a balanced way. We call this process as the procedure share.

We identify a most negative arc in $G(x)$ by associating a key to every node in N. We define in-key(i) of a node i as the reduced cost of a most negative incoming arc to node i in $G(x)$; if there is no negative incoming arc, then in-key(i) = 0. Let in-key(p) = min{in-key(i) : $i \in N$}. Thus, in-key(p) is the reduced cost of a most negative arc in $G(x)$. Note that for some node q, the arc (q, p) is a most negative arc in $G(x)$. Let $\varepsilon = -c^{\pi}_{qp}$ = -in-key(p). We refer to the node p as the node ***examined*** at the iteration. Once the node to be examined is identified, the algo-

rithmic description of the procedure share is given in Figure 3.3.

```
procedure share (x, π, p);
begin
     ε = -in-key(p);
     find the set of nodes S reachable from node p through
          eligible arcs in G(x);
     for every node i ∈ S do
          π(i) : = π(i) + ε/2;
end;
```

Figure 3.3 The procedure share

Updating potential π by the procedure share may create cycles in the admissible graph. We make the admissible graph again acyclic by repeatedly finding and cancelling admissible cycles. Then we find the node p such that a most negative arc is an incoming arc to node p. We call this process as the procedure cancel, whose algorithmic description is given in Figure 3.4.

```
procedure cancel (x, π. p);
begin
     repeatedly find and cancel admissible cycles until the
     admissible graph become acyclic;
     recompute in-key of every node and find the node p such that
     in-key(p) = min{in-key(i) : i ∈ N};
end;
```

Figure 3.4 The procedure cancel.

Our algorithm repeatedly performs the procedure share and the procedure cancel until the optimal solution is obtained. We first

describe the algorithm for integral arc costs, which is given in Figure 3.5.

```
algorithm cancel-and-share;
begin
    π: = 0 and x be any feasible flow;
    cancel (x, π, p);
    while in-key(p) ≤ -1/n do
    begin
        share (x, π, p);
        cancel (x, π, p);
    end;
end;
```

Figure 3.5 Cancel and share algorithm

Figure 3.6 illustrates the first iteration of the cancel-and-share algorithm for the flow on the example network given in Figure 3.1. Since $\pi = 0$, $c_{ij}^{\pi} = c_{ij}$ for every arc (i, j) $G(x)$ (See Figure 3.1(b)). The initial admissible graph consists of arcs (1, 3), (2, 4), (4, 1), (5, 1), and (5, 4), and is acyclic. Number shown in square brackets by the side of each node is the in-key. Since node 1 has the minimum in-key (note that (5, 1) is the most negative arc), it is the node to be examined at the first iteration (node 1 is enclosed by a square bracket). The procedure share finds the set $S = \{1, 3\}$ of nodes reachable from node 1 through eligible arcs, and increases the potentials of these nodes by -in-key (1)/2 = 9. Figure 3.6(a) shows the reduced costs after the update. The admissible network in Figure 3.6(a) is cyclic. Figure 3.6(b) shows the resultant residual network

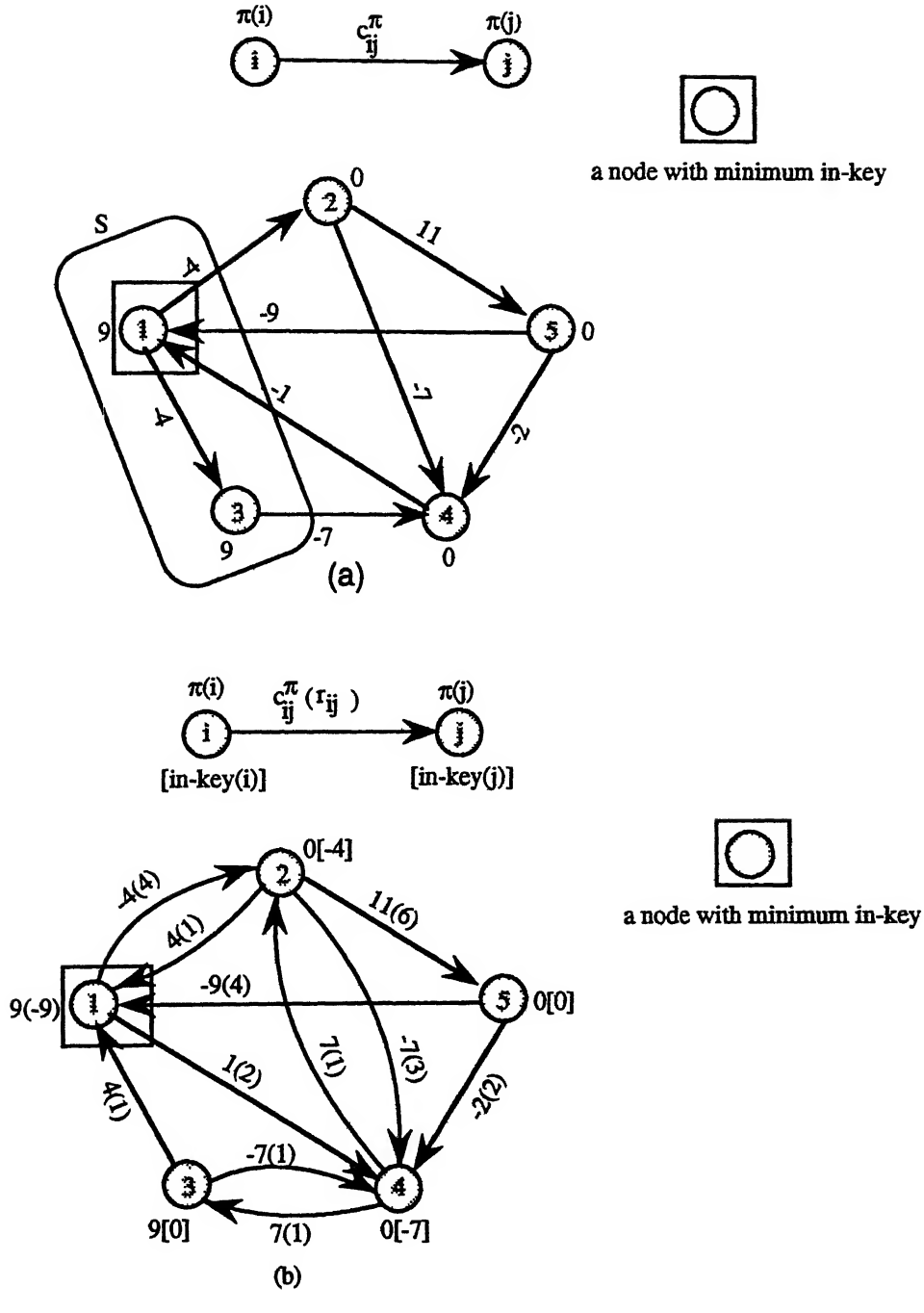


Figure 3.6 Illustration of an iteration of the algorithm cancel-and-share on Example 3.1.
(a) Procedure share .
(b) Procedure cancel

after the procedure cancel is applied. At the end of the iteration, node 1 is again selected for examination at the next iteration.

Analysis of the algorithm

Recall that in-key(p) of the node p at the beginning of an iteration is the reduced cost of a most negative arc. Thus, for ε = -in-key(p), the flow x is ε-optimal. At the termination of the algorithm, the flow x is ε-optimal for $\varepsilon < 1/n$ since in-key(p) > $-1/n$. By Lemma 3.1, the flow x is an optimal flow.

We need the following lemmas to bound the number of iterations.

Lemma 3.4 : *Cancelling admissible cycles does not decrease the in-key(i) of any node.*

Proof : Cancelling the flow on an admissible cycle adds only arcs with positive reduced costs to the admissible graph. Hence the lemma follows. ■

Lemma 3.5 : *Let node p be the node examined at an iteration. Then (a) at the end of the iteration, in-key(p) increases to at least half of its value at the beginning of iteration; and (b) if in-key(i) of any node i decreases, it decreases at least to in-key(p)/2.*

Proof : Let S denote the set of nodes reachable from node p through eligible arcs in $G(x)$. Updating the potential π in the procedure share increases the reduced cost of arcs in $(\bar{S}, S)$ (the backward arcs in the cut $[S, \bar{S}]$) by $\varepsilon/2$ where ε = -in-key(p).

Since the admissible graph is acyclic, the tail nodes of all incoming eligible arcs to node p lie in $\bar{S}$. For, if $q \in S$ for some eligible arc (q, p) then there is a directed path from p to q consisting of eligible arcs which forms a admissible cycle together with (q, p) which is a contradiction. This implies that all incoming eligible arcs are in the set $(\bar{S}, S)$. As a consequence, in-key(p) increases by $\varepsilon/2$.

Updating potentials decreases reduced costs of arcs in $(S, \bar{S})$ (the forward arcs in the cut $[S, \bar{S}]$) by $\varepsilon/2$. Since before the update $c_{ij}^{\pi} \geq 0$ for every arc (i, j) in $(S, \bar{S})$ by the construction of the set S, the reduced cost of any arc (i, j) is at least $-\varepsilon/2$. Consequently, keys of the nodes in $\bar{S}$ can only decrease, and if they do, they decrease at least to $-\varepsilon/2$.

Now the lemma follows from Lemma 3.4 and the fact that in-key(p) = $-\varepsilon$. ■

Lemma 3.6 : ***After at most n consecutive iterations of the algorithm, the reduced cost of the most negative arc in G(x) (the in-key of the node examined) increases to at least half of its original value.***

Proof : Let in-key(i_o) = $-\varepsilon$, where i_o is the node examined at the beginning of n consecutive iterations under consideration. Since in-key(i_o) is the reduced cost of the most negative arc, in-keys of all other nodes at least $-\varepsilon$.

Let Q denotes the set of nodes whose in-keys are less than $-\varepsilon/2$ at the beginning of these iterations. By Lemma 3.5(b), no node is added to Q after the first of these iterations. Further,

in-key of the node examined in the next iteration at least $-\varepsilon$. Repeating this argument for subsequent iterations, we conclude that (a) no node is added to Q in subsequent iterations; and (b) in-key of the node examined in subsequent iterations is non-decreasing. By Lemma 3.5(a), the in-key of the node examined increases to at least half of its original value at the beginning of an iteration; this implies, by conclusion (b), that at least one node is removed from Q at each iteration. Since Q consists of at most n nodes, after n consecutive iterations, Q will be empty. The lemma follows immediately. ■

Theorem 3.4 : *The algorithm cancel-and-share solves the minimum cost flow problem with integral costs in $O(n \log(nC))$ iterations.*

Proof : At the initial stage of the algorithm, since $\pi = 0$, the most negative arc in $G(x)$ has reduced cost at least $-C$ where C is the largest cost. By Lemma 3.6, the reduced cost of the most negative arc in $G(x)$ increases at least to half of its original value in at most n consecutive iterations. Hence, after $n(\lceil \log nC \rceil + 1) = O(n \log(nC))$ iterations, the reduced cost of the most negative arc is greater than or equal to $-C/2^{\lceil \log nC \rceil + 1} \geq -1/n$. Therefore, the resulting flow will be an $1/n$-optimal flow. From our discussion at the beginning of our analysis, this is an optimal flow.

A strongly polynomial implementation

The algorithm cancel-and-share runs in strongly polynomial number of iterations if we update potential π using a shortest path algorithm before cancel-and-share procedure at every iteration. We define $c^1_{ij} = \max(0, c^{\pi}_{ij})$ as the arc length of every

arc (i, j) in $G(x)$, and we find the shortest path distance $d(i)$ from node p being examined to every node $i \in N$. Then we update the potential π to π-d. If in-key$(p) < 0$ after update, share and cancel procedures are applied. We call the above updating procedure as adjust-potential, and the algorithm as enhanced cancel-and-share. The algorithmic description is given in Figure 3.7.

```
algorithm enhanced cancel-and-share;
begin
     π: = 0 and let x be any feasible flow;
     cancel (x, π, p);
     while in-key(p) < 0 do
     begin
          adjust-potential (x, π, p);
          share (x, π, p);
          cancel (x, π, p);
     end;
end;

Procedure adjust-potential (x, π, p);
begin
     find shortest path distance d(v) from node p to every
     node v ∈ N using arc lengths max (0, c_ij^π);
     π: = π-d;
     compute in-key(p);
end;
```

Figure 3.7 (a) The enhanced cancel-and-share algorithm.
(b) The procedure adjust-potential.

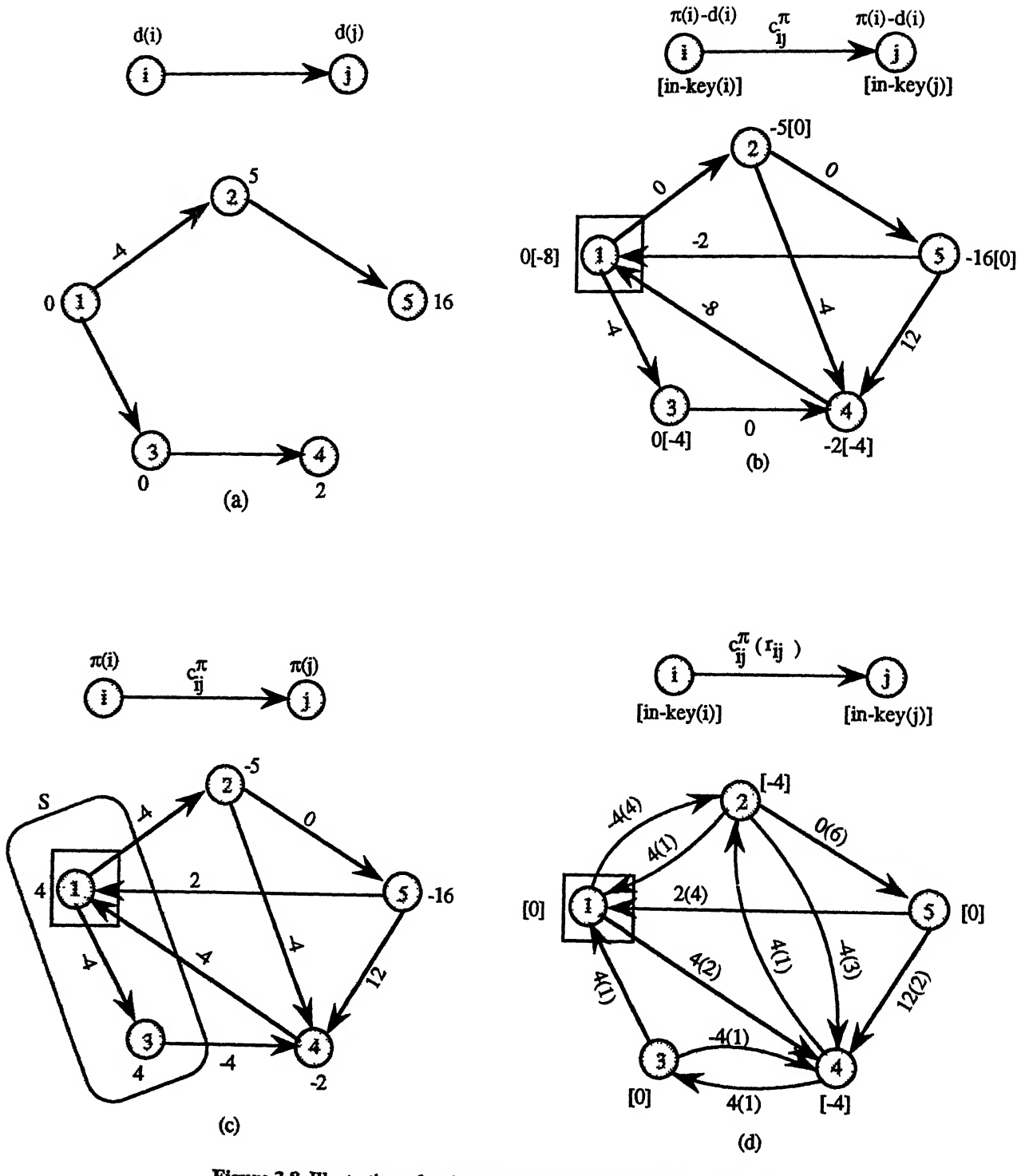


Figure 3.8 Illustration of an iteration of the algorithm enhanced cancel-and-share on Example 3.1.

(a) shortest distance using arc lengths $(0, c_{ij}^{\pi})$.

(b) reduced cost after the procedure adjust-potential.
(c) reduced cost after the procedure share
(d) The residual network after the procedure cancel.

Figure 3.8 illustrates the first iteration of the enhanced cancel-and-share algorithm for the flow on the example network given in Figure 3.1. Figure 3.8(a) gives shortest distances $d(.)$ from node 1 obtained using arc cost $(0, c_{ij}^{\pi})$. The procedure adjust-potentials updates the current potentials (which are zero) $\pi(.)$ to $\pi(.) - d(.)$. Updated potentials and the new reduced costs are shown in Figure 3.8(b). Since key(1) = -8 is still negative, we apply share and cancel subroutines. Figure 3.8(c) shows the potentials and reduced costs after the procedure share is applied. The set of nodes S enclosed by a rectangle is the set of nodes which are updated in the procedure share. Figure 3.8(d) shows the residual network after the procedure cancel is applied. Comparison with Figure 3.6 shows that adjust-potential, in effect, modify the reduced cost so that incoming eligible arcs to node 1 lie in a negative cycle whose cost is less than or equal to the reduced cost of the arc. For example, arc (5, 1) has reduced cost -2 which is the cost of the cycle 1-(1, 2)-2-(2, 5)-5-(5, 1)-1.

Lemma 3.7 : ***Updating potential by the procedure adjust-potential does not decrease the key of any node.***

Proof : Let $d(v)$ denotes the shortest distance from node p to node v using arc lengths $\max(0, c_{ij}^{\pi})$. By shortest path optimality conditions

$$\max(0, c_{ij}^{\pi}) + d(i) - d(j) \geq 0 \text{ for every arc } (i, j) \text{ in } G(x) \quad (3.4)$$

The statement (3.4) can be restated as the following two conditions.

$$c_{ij}^{\pi} + d(i) - d(j) \geq 0 \text{ for every arc } (i, j) \text{ in } G(x) \text{ with } c_{ij}^{\pi} \geq 0 \quad (3.5)$$

$$d(i) - d(j) \geq 0 \text{ for every arc } (i, j) \text{ in } G(x) \text{ with } c_{ij}^{\pi} < 0 \quad (3.6)$$

Since the new reduced cost after update are $c^{\pi}_{ij} + d(i) - d(j)$, the statement (3.5) implies that all non-eligible arcs remain non-eligible after the update; the statement (3.6) implies that the reduced cost of every eligible arc is non-decreasing. Therefore, the lemma follows from the definition of in-key. ■

The consequence of Lemma 3.7 is that the results in Lemma 3.5 and Lemma 3.6 also hold for the enhanced version of the algorithm. Further, we can replace in-key(p) in Lemma 3.5 by in-key(p) after the procedure adjust-potential.

We now consider two consecutive iterations of the algorithm enhanced cancel-and-share at which node i_o is examined. That is, in these iterations node i_o has a most negative arc as an incoming arc. Let the number of iterations between these two iterations be k-1. Let ρ_o, ρ_1, ρ_2, ..., ρ_{k-1}, ρ_k denote the in-keys of nodes examined at the beginning of these iterations, starting at the first iteration in which node i_o was examined and ending at the second iteration in which node i_o is again examined. Let $\bar{\rho}_o$, $\bar{\rho}_1$, ..., $\bar{\rho}_{k-1}$, $\bar{\rho}_k$ denote the in-keys of nodes examined in these iterations after the procedure adjust-potential in which we update the potentials by shortest path distances. By Lemma 3.7, $\rho_i \le \bar{\rho}_i$ for $0 \le i \le k$. By Lemma 3.5 (also see the remark at previous para), at the end of first iteration, in-key(i_o) increases at least to $\bar{\rho}_o/2$, and in subsequent k-1 iterations, if it decreases, it decreases at least to $\bar{\rho}_i/2$. Therefore, when node i_o is examined again, its in-key $\rho_k \ge \min\{\bar{\rho}_i/2 : 0 \le i \le k\}$ or $2\rho_k \ge \min\{\bar{\rho}_i : 0 \le i \le k\text{-}1\}$. But $\bar{\rho}_i = c^{\bar{\pi}}_{qp}$, where node p is the node examined at the corresponding iteration, and $\bar{\pi}$ is the potential and (q, p) the most negative incoming arc to node p after the

procedure adjust-potential applied. Consider the shortest path P_{pq} using arc length $c^1_{ij} = \max(0, c^{\pi}_{ij})$, where π is the potential before the procedure adjust-potential is applied. Since $\bar{\pi} = \pi - d$, where d represents the shortest path distances using arc lengths c^1_{ij}. Hence, for every arc (i, j) in P_{pq}, $c^1_{ij} + d(i) - d(j) = 0$ which implies $c^{\bar{\pi}}_{ij} \leq 0$. Therefore, the cost of the cycle $W := P_{pq} \cup \{(q,p)\}$, $C(W) = \sum_{(i,j)\in W} c_{ij} = \sum_{(i,j)\in W} c^{\bar{\pi}}_{ij} \leq c^{\bar{\pi}}_{qp} = \bar{\rho}_i$. Let $\bar{\rho} = \min\{\bar{\rho}_i : 0 \leq i \leq k-1\}$ and $\bar{W}$ denotes the corresponding cycle as defined above. Then, $c(\bar{W}) \leq \bar{\rho} \leq 2\rho_k$, where ρ_k is in-key(i_o) at the beginning of the second iteration at which node i_o is examined again.

In $(n+1)$ consecutive iterations of the algorithm, at least one node is examined twice. Let this node be node i_o in the above discussion. Thus at a stage in these $(n+1)$ consecutive iterations $G(x)$ has contained a cycle $\bar{W}$ such that $C(\bar{W}) \leq 2\rho_k$, where ρ_k is the key(i_o) when it is examined for second time. Since the key of the node examined is non-decreasing, ρ_k is less than or equal to the key of the node examined at the beginning of the $(n+1)$th iteration, which is by definition, the reduced cost of the most negative arc. Thus, we have proved the following lemma.

Lemma 3.8 : ***Let $-\varepsilon$ be the reduced cost of the most negative arc at the beginning of $(n+1)^{th}$ iteration of a set of $(n+1)$ consecutive iterations of the algorithm. Then, at one of these $(n+1)$ iterations, $G(x)$ has contained a cycle with cost at most -2ε.***

We define an arc to be ε-fixed if the flow on the arc is the same for all ε^1-optimal flows for $\varepsilon^1 \leq \varepsilon$. The following theorem is due to Tardos [1985] and Goldberg and Tarjan [1988].

Theorem 3.5 (Goldberg and Tarjan (1988)) : *Let $\varepsilon > 0$. Suppose a flow x is ε-optimal with respect to a potential π, and suppose that for some arc $(k, l) \in A$, $|c^{\pi}_{kl}| \geq 2n\varepsilon$, then arc (k, l) is ε-fixed.*

We apply the above theorem to prove the following result.

Theorem 3.6 : *Let x be a flow and suppose $G(x)$ contains a (directed) cycle W with mean cost $-\varepsilon$, $\varepsilon > 0$. Suppose a flow x^1 is ε^1-optimal and $2n\,\varepsilon^1 \leq \varepsilon$. Then, there is an arc $(k, l) \in W$ which ε^1-fixed.*

Proof : Let π denote potential for which the flow x^1 is ε^1-optimal, i.e., $c^{\pi}_{ij} \geq -\varepsilon^1$ for every arc (i, j) in $G(x^1)$. Since the mean cost of W is $-\varepsilon$, there must be at least one arc (k, l) in $G(x^1)$, which is either an arc or the reversal of an arc in W, having $|c^{\pi}_{kl}| \geq \varepsilon$. Since $2n\varepsilon^1 \leq \varepsilon$, we have $|c^{\pi}_{kl}| \geq 2n\varepsilon^1$. By Theorem 3.5 we obtain that arc (k, l) is ε^1-fixed.

Theorem 3.7 : *For the minimum cost flow problem with real valued costs, the algorithm enhanced cancel-and-share performs $O(mn \log n)$ iterations.*

Proof : Let $K = n\,2\lceil \log n \rceil + (n+1)$. We divide the iterations performed by the algorithm into groups of K consecutive iterations. We claim that each group of iterations fixes the flow on an additional arc. Since the algorithm can fix flows on at most m arcs, the algorithm performs at most $K = O(mn \log n)$ iterations.

Consider the first $(n+1)$ iterations in a group. Let $-\varepsilon$ denote the reduced cost of the most negative arc at the beginning

of $(n+1)^{th}$ iteration. By Lemma 3.8, at some stage of these $(n+1)$ iterations, $G(x)$ contained a cycle with cost at most -2ε. The mean cost of this cycle at most $-2\varepsilon/n = -\varepsilon^1$ (say). Each n consecutive iterations of the remaining iterations, by Lemma 3.6, increase the reduced cost of the most negative arc to at least half of its original value. Then, the reduced cost of the most negative arc at the end of this group $-\varepsilon^2 \geq \frac{-\varepsilon}{2^{2\lceil \log n \rceil}} \geq \frac{-\varepsilon}{2^{2 \log n}} = \frac{-\varepsilon}{n^2}$. Consequently, at the end of the group, we have a flow which is ε^2-optimal, and $\varepsilon^2 \leq \varepsilon/n^2 = \varepsilon^1/2n$, i.e., $2n\varepsilon^2 \leq \varepsilon^1$, where ε^1 is the mean cost of negative cycle in $G(x)$ at some stage of these iterations. Thus, by Theorem 3.6, at least one arc in this cycle is ε^2-fixed. Since the reduced cost of the most-negative arc is non-decreasing, the flow obtained in subsequent iterations are ε^3-optimal for $\varepsilon^3 \leq \varepsilon^2$. Hence the flow on this arc remains unchanged.

Complexity Analysis

The running time of each iteration is dominated by the time taken by the procedure cancel, which cancels admissible cycles until no admissible cycles are left. We use Goldberg and Tarjan's [1988] method to cancel admissible cycles. The method uses depth-first search. Each search advances only along eligible arcs. Whenever a search retreats from a node i, this node is marked as being on no admissible cycle. A search is allowed to visit only unmarked nodes. Whenever a search advances to a node it has already visited, an admissible cycle has been found. The cycle is cancelled and a new search begins. Goldberg and Tarjan [1988] describe two implementations of the above method. The

straightforward implementation runs in $O(nm)$ time. The implementation using a dynamic tree data structure (Sleator and Tarjan [1983, 1985], Tarjan [1983]) runs in $O(m \log n)$ time.

The procedure adjust-potential requires solving a shortest path problem which can be implemented in $O(m+n \log n)$ time (Fredman and Tarjan [1984]). Other operations in an iteration, updating potentials, updating the keys of nodes, and finding the node with minimum key can be done in $O(m)$ time. Thus we can implement each iteration in $O(m \log n) + O(m+n \log n) + O(m) = O(m \log n)$ time. By Theorem 3.4, the algorithm performs $O(n \log(nC))$ iterations for integral data. By Theorem 3.7, the algorithm performs $O(n\ m \log n)$ iterations for arbitrary real-valued costs. Thus we obtain the following theorem.

Theorem 3.8 : *The algorithm enhanced cancel-and-share solves the minimum-cost flow problem in $O(n\ m^2(\log n)^2)$ time for arbitrary real-valued arc costs, and in $O(nm \log n \min\{\log(nC), m \log n\})$ time for integral arc costs.*

Chapter 4

ARC TOLERANCES IN NETWORK FLOW PROBLEM

4.1 INTRODUCTION

Sensitivity analysis is a well-established area in operations research, in particular, for network and linear programming problems. The versatility of network programming needed the sensitivity analysis in its own right and a good body of research has been done in this direction. In this chapter, we address the sensitivity of cost functions associated with arcs in the network flow problems.

The sensitivity results provide useful information to the managerial decision making such as break-even selling price of a new product. Since there is always some uncertainty in data, it also provides a measure of robustness of optimal solution under small variation in data. Further, when the model appears as a subroutine in more complex problems, they provide useful information to perform the next step. The usefulness of sensitivity analysis can be gauged from the fact that every commercial network programming software, which may be a part of a linear programming system, provides sensitivity results as a part of standard output.

Hundreds of articles have been written on sensitivity analysis for linear programming. The book and the survey paper by Gal [1979, 1984] provides a comprehensive review on this topic. The sensitivity analysis is also an essential part in standard linear programming books, see, for example, Bradely et al. [1977] and Murthy [1976]. In linear programming, the sensitivity of a variable (activity) is estimated by the range in which cost coefficient (price) of the variable can vary, with all other cost coefficients unchanged, while keeping the current basis optimal. Values in this range are called as ***basic support prices*** (Ben-Israel et al. [1989]); we refer to this interval as ***basic tolerance interval***.

When the given optimal basis is degenerate, the range in which cost coefficient of a variable can vary, with all other cost coefficients unchanged, while keeping ***the current basic solution*** optimal is larger than the basic tolerance interval (which keeps ***the current basis*** optimal). We call this interval as ***the tolerance interval***. Values in this interval are known as support prices (Ben-Israel et al [1989]). The tolerance interval contains all basic tolerance intervals corresponding to the current basic solution; they coincide when the given basis is nondegenerate. However, tolerance intervals are not easier to calculate from the local data. We call the end points of a (basic) tolerance interval as (basic) tolerance limits or simply (basic) tolerances.

Srinivasan and Thompson [1972] specialized sensitivity results of linear programming to the transportation problem. They apply so called ***basis preserving cell operators*** to calculate basic

tolerances. Shier and Witzgal [1980] studied basic tolerances of an arc for the minimum cost flow problem (they call basic tolerances subtracted from the cost co-efficients as arc tolerances). It should be noted that the transportation problem and the minimum cost flow problem are equivalent models (see, for example, Ahuja et al. [1993]). In both the above articles, basic tolerance intervals of arcs are determined as follows. Let π be the node potential corresponding to the given optimal basis (which is a tree of the given network). Let (p, q) be a non-basic (non-tree) arc. If its flow is at its lower (upper) bound, then its basic tolerance interval is $[\pi(p)-\pi(q), \infty]$ ($[-\infty, \pi(p)-\pi(q)]$); this result is a straight forward application of the corresponding result for linear programming (see, for example, Murthy [1976]). Now consider the case of basic (tree) arc (p, q). Let $[S, \bar{S}]$ denote the cut obtained by deleting the arc (p, q) from the basis tree such that $p \in S$, $q \in \bar{S}$. Then the basic arc tolerance interval is $[c_{pq} - \varepsilon_2, c_{pq} + \varepsilon_1]$, where $\varepsilon_1(\varepsilon_2)$ is the minimum of absolute values of reduced costs of forward arcs at its upper (lower) bound and backward arcs at its lower (upper) bound in the set $[S, \bar{S}] - \{(p, q)\}$.

Shier and Witzgall [1980] also presented an algorithm to compute basic tolerance intervals for all arcs. Computation for non-tree arcs are straight forward and takes overall $O(m)$ time. They compute basic tolerance intervals for tree arcs as follows. They choose a leaf node (node with degree 1) in the basis tree; let (p, q) or (q, p) be the arc in the basis. The basic tolerance interval of this arc is computed by scanning the forward star $A(p)$ and backward star $B(p)$ of node p. Then this arc ((p, q) or (q, p))

is contracted and the process repeated for the basis tree in the contracted graph. The algorithm runs in $O(n^2)$ time. Using advanced data structures, Gusfield [1983] and Tarjan [1982] presented algorithms which run in $O(m \log n)$ and $O(m\ \alpha(m, n))$ time, respectively, where $\alpha(m, n)$ is a functional inverse of Ackermann's function (See Tarjan [1982]).

Ravi and Wendell [1988] have considered arc tolerances in which arc costs (as well as other data) are perturbed simultaneously and independently; the results are specialization of Wendell's [1985] results for linear programming. We do not consider simultaneous arc tolerances in this chapter.

Optimal basic (spanning tree) solutions to network flow solutions often have high degree of degeneracy. For example, dynamic networks for the vehicle allocation problem studied by Powell [1989] have been found to have 40-80% degenerate basic (tree) arcs (that is, flows on these arcs at are either upper or lower bounds). Further, most of the combinatorial algorithms do not produce ***basic*** optimal solutions. In view of the above facts, it is imperative to consider arc tolerances which preserves the optimality of a solution instead of basic arc tolerances. Fong and Srinivasan [1977] and Powell [1989] have studied the problem of determining nondegenerate shadow prices (node potentials and reduced costs of arcs).

Ahuja et al. [1993] treats unit change in an arc cost combinatorially which can be extended for more general changes. Rockafellar [1984] discuss sensitivity analysis for the separable convex cost flow problem.

In this chapter, we characterize arc tolerances by shortest non-singleton augmenting paths or, equivalently, shortest non-singleton path in the residual network. A non-singleton path is a path consisting of more than one arc. Extending the scope, we present our results for the more general separable convex cost flow problem; the analysis and the algorithm remain the same as that for the minimum cost flow problem. We present an algorithm which uses reoptimization to find shortest non-singleton paths between all pairs of nodes immediately giving tolerance intervals for all arcs. The algorithm runs in $O(n^3)$ time in array implementation, and in $O(nm + n^2 \log n)$ time in Fibonacci heap implementation. Compared to earlier results of Srinivasan and Thompson [1972] and Shier and Witzgall [1980], we compute tolerance intervals instead of basic tolerance intervals; our analysis can be applied to any optimal solution (not necessarily any basic optimal solutions) and to the more general separable convex cost flow problem.

The outline of the chapter is as follows.

In Section 4.2, we give the formulation for the separable convex cost flow problem and review some necessary definitions. Then, we present two well-known optimality conditions for the separable convex cost flow problem. Finally, we define the tolerance interval of an arc for the separable convex cost flow problem consistent with the definition of arc tolerance interval for the minimum cost flow problem. In Section 4.3, we present the theory which characterizes tolerance values by shortest non-singleton paths in the residual network. In Section 4.4, we present an algorithm to find shortest non-singleton paths between all pair of nodes.

4.2 THE BACKGROUND AND THE PROBLEM FORMULATION

In this section, we state the separable convex cost flow problem and formulate the problem to be studied in this chapter.

The separable convex cost flow problem is defined on a directed network, $G = (N, A)$. Here N is a set of nodes and A is a set of arcs. Each arc $(i, j) \in A$ has an associated real-valued convex cost function $C_{ij}(x_{ij})$ which gives the cost for the flow x_{ij} on arc (i, j). We also associate with each arc $(i, j) \in A$ a capacity u_{ij} (also called upper bound) that denotes the maximum amount of flow that can be sent on the arc and a lower bound l_{ij} that denotes the minimum amount of flow that must be sent on the arc. We associate with each node $i \in N$ a real number $b(i)$ representing its supply/demand. If $b(i) > 0$, node i is a supply node; if $b(i) < 0$, node i is a demand node with a demand of $-b(i)$; and if $b(i) = 0$, node i is a transshipment node. The decision variables in the separable convex cost flow problem are arc flows and we represent the flow on an arc $(i, j) \in A$ by x_{ij}. The separable convex cost flow problem (SCCF) is formulated as follows :

$$\text{Minimize } \sum C_{ij}(x_{ij}) \tag{4.1}$$

subject to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i) \text{ for all } i \in N, \tag{4.2}$$

$$l_{ij} \le x_{ij} \le u_{ij} \text{ for all } (i, j) \in A, \tag{4.3}$$

where $\sum_{i=1}^{A} b(i) = 0$. We denote the objective function by $C(x)$, i.e., $C(x) = \sum C_{ij}(x_{ij})$.

Marginal Costs

Consider an arc $(k, l) \in A$. Let $\bar{x}$ be a flow and $\bar{x}_{kl}$ denote the corresponding arc flow on arc (k, l). The right marginal cost of (k, l), with respect to the arc flow $\bar{x}_{kl}$, is the right derivative of the arc cost function $C_{kl}(x_{kl})$ at $\bar{x}_{kl}$, and we will denote it by $C^+_{kl}(x_{kl})$, or, simply by C^+_{kl}. Thus,

$$C^+_{kl}(\bar{x}_{kl}) = \underset{\delta \to 0}{\text{Limit}} \frac{C_{kl}(x_{kl}+\delta) - C_{kl}(x_{kl})}{\delta}, \quad \delta \geq 0 \qquad (4.4)$$

Similarly the left marginal cost of arc $(k. l)$ (with respect to $\bar{x}_{kl}$), denoted by $C^-_{kl}(\bar{x}_{kl})$ or C^-_{kl}, is the left derivative of $C_{kl}(x_{kl})$ at $\bar{x}_{kl}$, i.e.,

$$C^-_{kl}(\bar{x}_{kl}) = \underset{\delta \to 0}{\text{Limit}} \frac{C_{kl}(x_{kl}-\delta) - C_{kl}(x_{kl})}{-\delta}, \quad \delta \geq 0 \qquad (4.5)$$

Cost of Augmenting Cycles and Paths

Consider an augmenting cycle W with respect to a flow $\bar{x}$. The orientation of the cycle W is the direction along which the flow is augmented. Let $\overline{W}$ denote the set of arcs in W on which the flow is increased and $\underline{W}$ denote the set of arcs in W on which the flow is decreased. We define the cost $C(W)$ of the cycle W as the rate of change in the objective function as we augment the flow along the cycle W. Note that when we augment $\theta > 0$ units flow along the cycle W, the flow changes to $\bar{x}_{ij} + \theta$ on arcs in $\overline{W}$, and $\bar{x}_{ij} - \theta$ on arcs in $\underline{W}$; the flow remains the same on all other arcs. Thus,

$$C(W) = \underset{\theta \to 0}{\text{Limit}} \left[\sum_{(i,j) \in \overline{W}} \frac{C_{ij}(\overline{x}_{ij}+\theta) - C_{ij}(\overline{x}_{ij})}{\theta} + \sum_{(i,j) \in \underline{W}} \frac{C_{ij}(\overline{x}_{ij}-\theta) - C_{ij}(\overline{x}_{ij})}{\theta} \right].$$

From (4.4) and (4.5), we have

$$C(W) = \sum_{(i,j) \in \overline{W}} C^{+}_{ij}(\overline{x}_{ij}) - \sum_{(i,j) \in \underline{W}} C^{-}_{ij}(\overline{x}_{ij}). \tag{4.6}$$

Consider an augmenting path P with respect to the flow $\overline{x}$. Let $\overline{P}$ denote the set of arcs in P on which the flow is increased and $\underline{P}$ denote the set of arcs in P on which the flow is decreased. In a similar way the cost of an augmenting cycle is defined. We define the cost $C(P)$ of the path P as the rate of change in the objective function as we augment the flow along the path P. Thus,

$$C(P) = \sum_{(i,j) \in \overline{P}} C^{+}_{ij}(\overline{x}_{ij}) - \sum_{(i,j) \in \underline{P}} C^{-}_{ij}(\overline{x}_{ij}) \tag{4.7}$$

The Residual Network for Separable Convex Cost Flow Problems

Let $\overline{x}$ be a feasible flow to SCCF. The construction of the residual network $G = (N, A(\overline{x}))$ is similar to the construction of the residual network for the minimum cost flow problem except for arc cost. For every arc $(i, j) \in A$ with $\overline{x}_{ij} < u_{ij}$, we introduce an arc (i, j) in $A(\overline{x})$ with residual capacity $r_{ij} = u_{ij} - \overline{x}_{ij}$ and cost $\hat{c}_{ij} = C^{+}_{ij}(\overline{x}_{ij})$; for every arc $(i, j) \in A$ with $\overline{x}_{ij} > l_{ij}$, we introduce an arc (j, i) with $r_{ji} = \overline{x}_{ij}$ and cost $\hat{c}_{ji} = -C^{-}_{ij}(\overline{x}_{ij})$. Let

$$A(\overline{x}) = A^{+}(\overline{x}) \cup A^{-}(\overline{x}), \tag{4.8}$$

where

$$A^+(\bar{x}) = \{(i,j) : (i, j) \in A \text{ and } \bar{x}_{ij} < u_{ij}\}, \tag{4.9}$$

$$A^-(\bar{x}) = \{(i,j) : (j, i) \in A \text{ and } \bar{x}_{ji} > l_{ji}\}. \tag{4.10}$$

Further,

$$r_{ij} = \begin{cases} u_{ij} - \bar{x}_{ij} & \text{if } (i, j) \in A^+(\bar{x}) \\ \bar{x}_{ji} & \text{if } (i, j) \in A^-(\bar{x}) \end{cases} \tag{4.11}$$

$$\hat{c}_{ij} = \begin{cases} c^+_{ij}(\bar{x}_{ij}) & \text{if } (i, j) \in A^+(\bar{x}) \\ -c^-_{ji}(\bar{x}_{ji}) & \text{if } (i, j) \in A^-(\bar{x}) \end{cases} \tag{4.12}$$

The network $G(\bar{x})$ contains cycles of the type i - (i, j) - j - (j, i) - i : = $(i, j) \cup (j, i)$, if the flow $\bar{x}_{ij}$ lies in between the bounds, i.e., $l_{ij} < \bar{x}_{ij} < u_{ij}$. We call these cycles as trivial cycles, since these cycles represent zero flow augmentation on the corresponding arcs. Recall that, for the minimum cost flow problem, $\hat{c}_{ij} = c_{ij}$ for $(i, j) \in A^+(\bar{x})$ and $\hat{c}_{ij} = -c_{ji}$ for $(i, j) \in A^-(\bar{x})$.

We note that any augmenting cycle W^1 with respect to the flow $\bar{x}$ corresponds to a directed cycle W in $G(\bar{x})$; the directed cycle W is a non-trivial cycle (i.e., is not a trivial cycle), and is obtained by reversing the arcs in W^1 on which the flow is decreased. Conversely, any non-trivial directed cycle W in $G(\bar{x})$ corresponds to an augmenting cycle W^1 in G. The cost of a directed cycle W in $G(\bar{x})$ is defined by

$$C(W) = \sum_{(i,j) \in W} \hat{c}_{ij} \tag{4.13}$$

Thus, by construction, W and its corresponding cycle W^1 have the same cost, i.e., $C(W) = C(W^1)$. Henceforth we mainly work with the

residual network, since it simplifies the discussion and it is equivalent to working with the original network.

4.2.1 Optimality Conditions

We now present two optimality conditions, which are necessary for our analysis. These results are well-known, (for example, see Rockafellar [1984]), but for the sake of completeness, we give the proofs in our terminology. See also Iri [1969] for related results.

Let x be a feasible flow. For any augmenting cycle W with respect to the flow x, we define its incidence vector $e_W \in R^m$ as the vector having 1 for the co-ordinates corresponding to the arcs in $\overline{W}$, -1 for the co-ordinates corresponding to the arcs in $\underline{W}$, and 0 for others. Then $x + \theta\, e_W$ is the flow obtained by augmenting θ units along the cycle W. Thus, by definition

$$C(W) = \lim_{\theta \to 0} \frac{c(x + \theta\, e_W) - c(x)}{\theta}, \quad \theta \geq 0. \tag{4.14}$$

Let y be any other flow. Then y-x is a feasible circulation in $G(x)$. By the flow decomposition theorem (see, for example, Ahuja et al. [1993]), y-x can be decomposed into at most m cyclic flows on $G(x)$. That is, the flow y can be obtained from the flow x by at most m cyclic augmentations. Hence, for some $\gamma \leq m$, $\theta_k > 0$ for every $1 \leq k \leq \gamma$, and

$$y\text{-}x = \sum_{k=1}^{\gamma} \theta_k\, e_{W_k} \tag{4.15}$$

We define a vector $C^1 \in R^m$ as follows.

$$C^1_{ij} = \begin{cases} C^+_{ij}(x_{ij}) & \text{if } y_{ij} > x_{ij} \\ C^-_{ij}(x_{ij}) & \text{if } y_{ij} < x_{ij} \\ 0 & \text{otherwise .} \end{cases}$$

Then

$$C^1(y\text{-}x) = \sum_{k=1}^{\gamma} \theta_k \, C(W_k) . \tag{4.16}$$

From the convexity of the function $C_{ij}(x_{ij})$ (see Jan van Tiel [1984, Theorem 1.6]; also see Rockafellar [1970, Chapter 23]), we have

$$C_{ij}(y_{ij}) \geq C_{ij}(x_{ij}) + C^+_{ij}(x_{ij}) \, (y_{ij}\text{-}x_{ij}) , \tag{4.17}$$

$$C_{ij}(y_{ij}) \geq C_{ij}(x_{ij}) + C^-_{ij}(x_{ij}) \, (y_{ij}\text{-}x_{ij}) . \tag{4.18}$$

Adding the left hand side and the right hand side of the inequalities (4.17) corresponding to the arcs (i, j) with $y_{ij} > x_{ij}$ and of the inequalities (4.18) corresponding to the arcs (i, j) with $y_{ij} < x_{ij}$, and adding the terms $C_{ij}(y_{ij})$ and $C_{ij}(x_{ij})$ for the arcs (i, j) with $y_{ij} = x_{ij}$ to the resultant right hand and left hand sums respectively, we have

$$C(y) \geq C(x) + C^1(y\text{-}x). \tag{4.19}$$

Thus, by (4.16), we have

$$C(y) \geq C(x) + \sum_{k=1}^{\gamma} \theta_k \, C(W_k).$$

We are now in a position to prove the following optimality conditions.

Theorem 4.1 (Negative Cycle Optimality Conditions) : ***A feasible flow x to SCCF is optimal if and only if G contains no negative augmenting cycle W with respect to the flow x (equivalently, the residual network G(x) contains no negative cycle).***

Proof. Necessity : Let W be an augmenting cycle with respect to the flow x. Then $x + \theta\, e_W$ is a feasible flow for some $\theta > 0$, and for all $\theta > 0$ such that $x + \theta\, e_W$ is feasible. By the optimality of the flow x, we have

$$C(x + \theta\, e_W) \geq C(x).$$

Consequently, $\lim_{\theta \to 0} \dfrac{C(x + \theta\, e_W) - C(x)}{\theta} \geq 0.$

Therefore, by (4.13), $c(W) \geq 0$.

Sufficiency : Let y be any arbitrary feasible flow and $C(w) \geq 0$ for any augmenting cycle W. Then, from (4.15) and (4.19), there exists augmenting cycles $W_1, \ldots, W_r$, and $\theta_i \geq 0$, for all $1 \leq i \leq \gamma \leq m$ such that

$$y = x + \sum_{k=1}^{\gamma} \theta_k\, e_{W_k},$$

and

$$C(y) \geq C(x) + \sum_{k=1}^{\gamma} \theta_k\, C(W_k).$$

By assumption, $C(W_k) \geq 0$. Since $\theta_k > 0$, we have $C(y) \geq C(x)$. Hence x is an optimal flow. ■

The optimality conditions in Theorem 4.1 are equivalent to the optimality conditions given in Rockafellar [1984, pp. 435]. We derive the following optimality conditions, which are equivalent to the optimality conditions given in Rockafellar [1984, pp 382], using Theorem 4.1.

Theorem 4.2 (Reduced Cost Optimality Conditions) : *A feasible flow x to (SCCF) is optimal if and only if there exists a potential π satisfying the following conditions :*

$$\hat{c}^{\pi}_{ij} \geq 0 \text{ for every arc } (i, j) \text{ in } G(x), \qquad (4.20)$$

where $\hat{c}_{ij}$ *are the costs as defined by (4.12), and*

$$\hat{c}^{\pi}_{ij} = \hat{c}_{ij} - \pi(i) + \pi(j)$$

Proof : The sufficiency of the conditions (4.20) is straight forward. Note that, for any directed cycle W in $G(x)$,

$$C(W) = \sum_{(i,j)\in W} \hat{c}_{ij} = \sum_{(i,j)\in W} \hat{c}^{\pi}_{ij} ,$$

since $\pi(i)$'s in the second sum cancel out. Consequently, the conditions in (4.20) imply $C(W) \geq 0$ for any cycle W in $G(x)$. Hence, by Theorem 4.1, the flow x is optimal.

Necessity. We augment the network $G(x)$ to $\hat{G}(x)$ by adding node s and connecting the node s to every node $i \in N$ by an arc (s, i) with cost 0. In the network $\hat{G}(x)$, every node i has a directed path from s. Further, by assumption that x is optimal, the network contains no negative cycle (Theorem 4.1); consequently, $\hat{G}(x)$ contains no negative cycle since the new arcs do not introduce any directed cycle. The above properties imply that the shortest path distance $d(i)$ for each node $i \in N$ from node s exists and it is finite (see Ahuja et al. [1993, Sections 5.2 and 5.3]). Further, they satisfy the following conditions:

$$d(j) \leq d(i) + \hat{c}_{ij} \text{ for every } (i, j) \text{ in } \hat{G}(x). \qquad (4.21)$$

Let $\pi = -d$. Then $\hat{c}^{\pi}_{ij} \geq 0$ for every (i, j) in $\hat{G}(x)$. Since $G(x)$ is a subgraph of $\hat{G}(x)$, the necessity follows. ■

Note that, by construction of $G(x)$, $G(x)$ contains an arc (i, j) with cost $C_{ij}(x_{ij})$ for every arc $(i, j) \in A$ with $x_{ij} < u_{ij}$, and

an arc (j, i) with cost $-C^-_{ij}(x_{ij})$ for every arc $(i, j) \in A$ with $x_{ij} > l_{ij}$. (Henceforth we will assume, without loss of generality, lower bounds l_{ij}s are zero). Therefore, the reduced cost optimality conditions are the same as the following optimality conditions:

$$\pi(i) - \pi(j) \le C^+_{ij}(x_{ij}) \qquad \text{for every arc } (i, j) \in A \text{ with } x_{ij} = 0. \tag{4.22a}$$

$$C^-_{ij}(x_{ij}) \le \pi(i) - \pi(j) \le C^+_{ij}(x_{ij}) \qquad \text{for every arc } (i, j) \in A \text{ with } 0 < x_{ij} < u_{ij}. \tag{4.22b}$$

$$C^-_{ij}(x_{ij}) \le \pi(i) - \pi(j) \qquad \forall\ (i, j) \in A \text{ with } x_{ij} = u_{ij}. \tag{4.22c}$$

4.2.2 Arc Tolerance Analysis

We investigate the following problem in this chapter. How much the cost function associated with an arc can be perturbed without changing the arc flows in a given optimal flow to SCCF. We will perturb the cost functions of an arc one at a time, holding all other cost functions, supplies/demands and capacities constant. We refer to this problem as ***Arc Tolerance Analysis***. In this section, we develop a combinatorial theory for arc tolerance analysis to SCCF, which is based on the negative cycle optimality conditions given in Theorem 4.1.

We call a real-valued convex function on R as a ***valid perturbation*** of the convex function associated with an arc in SCCF if the given optimal flow remains optimal when we replace the latter by the former in the objective function. To be more precise, let $\bar{x}$ be an optimal solution to the original problem and (k, l) and $C_{kl}(x_{kl})$ be the arc under consideration and its cost

function, respectively. Then, a real valued convex function $\tilde{C}_{kl}(x_{kl})$ is a valid perturbation of $C_{kl}(x_{kl})$ if $\bar{x}$ remains optimal when we replace $C_{kl}(x_{kl})$ by $\tilde{C}_{kl}(x_{kl})$ in the objective function.

It is clear from the construction of the residual network $G(\bar{x})$ and the negative cycle optimality conditions that the factors associated with the arc (k, l) which influence the optimality of the flow $\bar{x}$ are only the marginal costs of $C_{kl}(x_{kl})$ at $\bar{x}_{kl}$. In the next section, we show that there is a closed interval $[\alpha_{kl}, \beta_{kl}]$ such that a convex function $\tilde{C}_{kl}(x_{kl})$ is a valid perturbation of $C_{kl}(x_{kl})$ if and only if at least one of the marginal costs of $\tilde{C}_{kl}(x_{kl})$ at $\bar{x}_{kl}$ falls in the interval $[\alpha_{kl}, \beta_{kl}]$. We call this interval as the tolerance interval of the arc (k, l) in SCCF.

The tolerance interval of an arc in SCCF, by definition, characterize effectively all the desired perturbations of the cost function associated with the arc. In MCF, the desired perturbations are restricted to linear functions. In this case, $C^{+}_{kl}(x_{kl}) = C^{-}_{kl}(x_{kl}) = c_{kl}$, and the tolerance interval for (k, l) in MCF is the interval in which cost coefficient c_{kl} can vary, while keeping the optimality of a given solution.

4.3 THEORY

In this section, we develop a theory to evaluate the tolerance interval for each arc in A for a given optimal flow. In the following discussions in this chapter, we denote the given optimal flow to SCCF by $\bar{x}$ and its residual network by $G(\bar{x})$.

Let (k, l) denote an arbitrary but fixed arc which is considered for finding the tolerance interval. Let $\tilde{C}_{kl}(x_{kl})$ be

any convex function replacing $C_{kl}(x_{kl})$. Thus, the new perturbed cost function is

$$\tilde{C}(x) = \sum_{\substack{(i,j)\in A \\ (i,j)\neq(k,l)}} C_{ij}(x_{ij}) + \tilde{C}_{kl}(x_{kl}).$$

By definition, $\tilde{C}_{kl}(x_{kl})$ is a valid perturbation if the flow $\bar{x}$ is optimal with respect to $\tilde{C}(x)$. Let $\tilde{G}$ be the network obtained from $G(\bar{x})$ after deleting the arcs (k, l) and (l, k) if they are present in $G(\bar{x})$. From the construction of $G(\bar{x})$, we have

$$G(\bar{x}) = \begin{cases} \tilde{G} \cup \{(l, k)\}, & \text{if } \bar{x}_{kl} = 0; \\ \tilde{G} \cup \{(k, l)\}, & \text{if } \bar{x}_{kl} = u_{kl}; \\ \tilde{G} \cup \{(k, l), (l, k)\} & \text{if } 0 < \bar{x}_{kl} < u_{kl}. \end{cases}$$

By the negative cycle optimality conditions, $G(\bar{x})$ contains no negative cycles. Further, after the perturbation, the cost of arcs (k, l) and (l, k) in $G(\bar{x})$ change to $\tilde{C}_{kl}^{+}(\bar{x}_{kl})$ and $-\tilde{C}_{kl}^{-}(\bar{x}_{kl})$ respectively, costs of all other arcs remain intact. These two facts lead to the following observations.

Property 4.1 : (a) $\tilde{G}$ contains no negative cycle. (b) If, after the perturbation, $G(\bar{x})$ contains a negative cycle, then the cycle consists of either arc (k, l) or arc (l, k).

If a minimum cost cycle containing an arc is non-negative, then any cycle containing the same arc is non-negative. Thus Property 4.1(b) implies the following property.

Property 4.2 : After the perturbation, $G(\bar{x})$ contains no negative cycles if and only if the following statements hold : (a) Any minimum cost cycle consisting of arc (k, l) in $G(\bar{x})$ has

and only if $\tilde{C}^{+}_{kl}(\bar{x}_{kl}) + \tilde{d}_l(k) \geq 0$ if (k, l) in $G(\bar{x})$, and $-\tilde{C}^{-}_{kl}(\bar{x}_{kl}) + \tilde{d}_k(l) \geq 0$ if (l, k) is in $G(\bar{x})$. But (k, l) is in $G(\bar{x})$ if $\bar{x}_{kl} < u_{kl}$, and (l, k) is in $G(\bar{x})$ if $\bar{x}_{kl} > 0$. Hence the lemma. ■

Note that a shortest path from node k to node l in $\tilde{G}$ is a shortest path from node k to node l excluding the single arc path k-(k,l)-l in $G(\bar{x})$, which in turn is a shortest augmenting path from node k to node l in G excluding the path k-(k,l)-l or the path k-(l,k)-l. For this reason we call the distance $\tilde{d}_k(l)$ as the non-singleton shortest augmenting distance from node k to node l. Recall that the tolerance interval $[\alpha_{kl}, \beta_{kl}]$ of arc (k, l) is, by definition, the interval with the property that a convex function $\tilde{C}_{kl}(x_{kl})$ is a valid perturbation if and only if at least one of $\tilde{C}^{-}_{kl}(\bar{x}_{kl})$ and $\tilde{C}^{+}_{kl}(\bar{x}_{kl})$ lies in this interval. The following theorem gives the tolerance interval.

Theorem 4.3 : *Let $\tilde{d}_k(l)$ and $\tilde{d}_l(k)$ denote the non-singleton shortest augmenting distances with respect to the flow $\bar{x}$ (or equivalently, the non-singleton shortest distances in $G(\bar{x})$) from node k to node l and node l to node k respectively. Then the tolerance interval $[\alpha_{kl}, \beta_{kl}]$ of arc (k, l) is given below :*

$$[\alpha_{kl}, \beta_{kl}] = \begin{cases} [-\tilde{d}_l(k), \infty), & \text{if } \bar{x}_{kl} = 0; \\ (-\infty, \tilde{d}_k(l)], & \text{if } \bar{x}_{kl} = u_{kl}\ ; \\ [-\tilde{d}_l(k), \tilde{d}_k(l)], & \text{if } 0 < \bar{x}_{kl} < u_{kl}\ . \end{cases} \tag{4.23}$$

Proof : Since, by Property 4.3, $-\tilde{d}_l(k) \leq \tilde{d}_k(l)$, $[\alpha_{kl}, \beta_{kl}]$ is non empty in all three cases in (4.23). Further, by convexity of $\tilde{C}_{kl}(x_{kl})$, one has

$$\tilde{C}^{-}_{kl}(x_{kl}) \leq \tilde{C}^{+}_{kl}(x_{kl}) \tag{4.24}$$

Sufficiency : When both $\tilde{c}^{-}_{kl}(\bar{x}_{kl})$ and $\tilde{c}^{+}_{kl}(\bar{x}_{kl})$ belong to $[\alpha_{kl}, \beta_{kl}]$, the conditions (a) and (b) in Lemma 4.1 are satisfied in all three cases, namely, $\bar{x}_{kl} = 0$, $\bar{x}_{kl} = u_{kl}$ and $0 < \bar{x}_{kl} < u_{kl}$. Hence $\tilde{c}_{kl}(\bar{x}_{kl})$ is a valid perturbation. Suppose only $\tilde{c}^{-}_{kl}(\bar{x}_{kl})$ belongs to $[\alpha_{kl}, \beta_{kl}]$. Then $-\tilde{d}_l(k) \leq \tilde{c}^{-}_{kl}(\bar{x}_{kl}) \leq \tilde{d}_k(l)$. But the inequality (4.24) implies that $\tilde{c}^{+}_{kl}(\bar{x}_{kl}) \geq -\tilde{d}_l(k)$ for the case $0 < \bar{x}_{kl} < u_{kl}$; hence $\tilde{c}_{kl}(x_{kl})$ is a valid perturbation. The cases $\bar{x}_{kl} = 0$, $\bar{x}_{kl} = u_{kl}$ are straight-forward. The arguments for the other possibility that only $\tilde{c}^{+}_{kl}(\bar{x}_{kl})$ belongs to $[\alpha_{kl}, \beta_{kl}]$ are similar.

Necessity : When both $\tilde{c}^{-}_{kl}(\bar{x}_{kl})$ and $\tilde{c}^{+}_{kl}(\bar{x}_{kl})$ lie outside $[\alpha_{kl}, \beta_{kl}]$, at least one of the conditions in Lemma 4.1 is violated. Therefore, $\tilde{c}_{kl}(x_{kl})$ is not a valid perturbation. ■

4.4 ALGORITHM FOR FINDING NON-SINGLETON SHORTEST PATHS

In Section 4.3, we have shown that the tolerance interval of an arc (k, l) can be immediately calculated, once we know non-singleton shortest distances $\tilde{d}_k(l)$ and $\tilde{d}_l(k)$ in $G(\bar{x})$ (Theorem 4.3). Since $G(\bar{x})$ contains no negative cycles, but contains usually many negative arc costs, we can find tolerance interval for all arcs by applying a label correcting shortest path algorithm (such as Bellman and Ford algorithm) $O(m)$ times. The best (strongly) polynomial time bound for any label correcting algorithm is $O(nm)$ [see, for example, Ahuja et al. [1993] for more details]. This straight-forward method is highly inefficient, since the overall complexity for computing all relevant non-singleton shortest distances is $O(nm^2)$. In this section, we improve this complexity significantly in two ways :

(i) By working with an optimal node potential.

(ii) By reoptimization.

Working with an Optimal Node Potential

The reduced cost optimality conditions (Theorem 4.2) guarantee that there exists a node potential π such that $\hat{c}^{\pi}_{ij} \geq 0$ for all arcs in $G(\bar{x})$. We call such a potential as an optimal potential. The following property shows that working with costs $\hat{c}^{\pi}_{ij}$ is equivalent to working with $\hat{c}_{ij}$ (recall that we use the notation $\hat{c}_{ij}$ to denote the cost of arc (i, j) in $G(\bar{x})$).

Property 4.4 : For any (directed) path P from node k to node l,

$$\sum_{(i,j)\in P} \hat{c}^{\pi}_{ij} = \sum \hat{c}_{ij} - (\pi(k) - \pi(l))$$

Proof : The summation $\hat{c}^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j)$ over arcs in P cancels out all $\pi(i)$'s except $\pi(l)$ and $-\pi(k)$. ■

Property 4.4 implies that one could work with $\hat{c}^{\pi}_{ij}$'s to find shortest distance from node k to node l and then add $\pi(k) - \pi(l)$ to get the shortest distance with respect to the original costs $\hat{c}_{ij}$. Thus, if we denote the non-singleton shortest distance from node p to node q by $\tilde{d}^{\pi}_{p}(q)$, then

$$\tilde{d}_{p}(q) = \tilde{d}^{\pi}_{p}(q) + \pi(p) - \pi(q) \qquad (4.25)$$

We often get a node potential π satisfying $\hat{c}^{\pi}_{ij} \geq 0$ for every (i, j) in $G(\bar{x})$, as a by-product of solving SCCF. Otherwise, we use the procedure applied in Theorem 4.2 to find such a node potential. The procedure has the following three steps.

(i) Augment the network $G(\bar{x})$ by adding a node s to $G(\bar{x})$ and connect this node to every node i by an arc (s, i) of cost 0.

(ii) Find shortest distance $d(i)$ from node s to every node i in the augmented network by applying a label correcting shortest path algorithm.

(iii) Let $\pi = -d$.

The step (ii) dominates all other steps. This step requires applying a label-correcting shortest path algorithm which, as we pointed out at the beginning of this section, requires $O(nm)$ time. We call this procedure as procedure potential-initialization.

Finding a Shortest Path Tree

Once we have optimal potentials at hand, we proceed as follows. We choose a node p and apply Dijkstra's algorithm to find a shortest tree T_p from node p in $G(\bar{x})$ using arc costs $\hat{c}_{ij}^{\pi}$. The path from node p to any node i in T_p is a shortest path from node p to node i, and we denote the corresponding shortest distance by $d_p^{\pi}(i)$.

Dijkstra's algorithm maintains at any intermediate step, a directed out-tree T at the (root) node p, spanning a set of node S for which the shortest distances are already found. The labels $d(i)$ for nodes in $\bar{S} = N\text{-}S$ give the distance of a shortest path to node i passing through nodes only in S. A node i with minimum label in $\bar{S}$ is chosen and added to S and labels of nodes adjacent to j are updated. Since we use this iterative process for reoptimization, we give its algorithmic description in Figure 4.1.

```
procedure Dijkstra's;
begin
    while |S| < |N| do
    begin
        let i ∈ S̄ be a node for which d_p^π(i) = min{d_p^π(j): j ∈ S̄}
        S = S u{i};
        S̄ = S - {i};
        for each (i, j) ∈ A(i) do
            if d_p^π(j) > d_p^π(i) + c_ij^π then d_p^π(j) := d_p^π(i) + c_ij
                                    and pred(j) = i;
    end;
end;
```

Figure 4.1

At the end of the algorithm, for each node i, the predecessor index of node i, $pred(i)$ gives the predecessor node j of node i in the final shortest path tree; $pred(p) = 0$.

Constructing Tree Indices of T_p

Using indices $pred(i)$'s we construct other tree indices, namely, depth index and thread index of each node $i \in N$, denoted by $depth(i)$ and $thread(i)$ (see Section 2.2). First we build adjacency list of each node i in T_p, denoted by SUCC(i), by examining $pred(i)$ of each node (i). Then applying breath-first and depth-first search in T_p using SUCC(i)'s, we construct depth indices and thread indices, respectively. The procedure takes $O(n)$ time. For more details, see Ahuja et al. [1993]. We shall see below that these indices are needed for efficient reoptimization.

The Reoptimization Procedure

We first point out that the tree path in T_p from p to all nodes except for those in SUCC(p) are non-singleton shortest paths. To find a non-singleton shortest path from p to node q in SUCC(p), we delete the arc (p, q) from T_p as well as from $G(\bar{x})$; as a result all relevant information, $pred(i)$'s and $d_p(i)$'s, become invalid only for nodes in $D(q)$, the set of all descendants of node q. We regain this information by applying Dijkstra's algorithm starting at the stage where only nodes in $D(q)$ are temporarily labeled. Since we need the data of the tree T_p for applying this procedure to other nodes in SUCC(p), we will not disturb these data; we will store the new distances in a new array, $nsd(i)$. If we need actual path, we will also store the new predecessor index as $nspred(i)$ for $i \in D(q)$ such that $q \in SUCC(p)$. Note that, for each node i, $nsd(i)$ denotes the shortest distance from node p to node i when arc (p, q) is deleted.

We initialize the label $nsd(j)$ for each node $j \in D(q)$ as follows :

$$nsd(j) = \min \{d_p^{\pi}(i) + c_{ij}^{\pi} : (i, j) \in B(j) : i \in N\text{-}D(q)\}$$

where $B(j)$ is the backward star of node j.

To perform this process efficiently, we first mark nodes in $D(q)$; we do this by tracing $thread(i)$ starting from node q until the depth of node next traced at least is equal to that of node q (we use depth indices). Then, we again visit nodes in $D(q)$ one by one using thread indices, and scan their backward star to initialize $nsd(j)$ for each $j \in D(q)$. The algorithmic description of this

procedure, named setup-potentials($D(q)$) is given in Figure 4.2.

```
procedure setup-potentials(D(q));
begin
    mark(q) := q ; nsd(q) = ∞ ;
    k := thread(q);
    while depth(k) < depth(q) do
    mark(k) = q, nsd(k) := ∞ and k := thread(k);
    for each (i, q) ∈ B(q) do
      if mark(i) ≠ q and nsd(q) > d_p^π(i) + c_ij^π then
         nsd(q) := d_p^π(i) + c_ij^π and nspred(q) := i;
    k = thread(q);
    while depth(k) < depth(q) do
    begin
         for each (i, k) ∈ B(k) do
         begin
              if mark(i) ≠ q then
              if nsd(k) > d_p^π(i) + c_ik^π then
              nsd(k) := d_p^π(i) + c_ij^π and nspred(k) := i;
         end;
         k := thread(k);
    end;
end;
```

Figure 4.2

We now apply Dijkstra's algorithm's iterative loop (see Figure 4.1) taking $S = N\text{-}D(q)$ as the initial set of permanently labelled nodes; for $i \in D(q)$, we use the label $nsd(i)$ instead of $d_p^\pi(i)$. At the termination of the iterative loop, $nsd(q)$ is the non-singleton shortest distance from p with respect to costs c_{ij}^π.

The justification follows from the correctness of Dijkstra's algorithm. When the arc (q, p) is deleted, the shortest distance $d_p^{\pi}(i)$ for each node $i \in N\text{-}D(q) = S$ remains the same. The procedure setup-potentials initializes the distance labels $nsd(i)$ to the shortest distance to node i passing through only nodes in $S = N\text{-}D(q)$ in the graph $G(\bar{x}) - \{(p, q)\}$, for each node $i \in D(q)$. Thus we can apply Dijkstra's algorithm taking $S = N\text{-}D(q)$ as the initial set of permanently labeled nodes. Hence, at the termination of reoptimization process, $nsd(q)$ is the shortest distance in $G(\bar{x})\text{-}\{(p\text{-}q)\}$, that is, the non-singleton shortest distance from node p to node q.

We now analyze the complexity of the reoptimization process. We refer to the reoptimization process done after deleting the arc (p, q) as reoptimization((p, q)). It has two steps :

(i) setting initial potentials for nodes in $D(q)$;

(ii) applying Dijkstra's algorithm on the graph $G(x) - \{(p, q)\}$ taking $S = N\text{-}D(q)$ as the initial set of permanently labeled nodes.

For step (i), marking nodes in $D(q)$ takes $|D(q)|$ time; scanning the backward star of each node in $D(q)$ takes $O\left(\sum_{i \in D(q)} |B(i)|\right)$ time. Thus, the complexity of step (i) is $O\left(\sum_{i \in D(q)} |B(i)|\right)$ time. In step (ii), scanning forward stars of nodes in $D(q)$ takes $O\left(\sum_{i \in D(q)} |F(i)|\right)$ time. Selecting the node with the minimum distance label takes $O(|D(q)|)$ time, if we store $nsd(i)$'s of nodes in $D(q)$ in an array (we can use thread indices to scan the nodes in $D(q)$ only); hence, this takes $O(|D(q)|^2)$ in

total. Alternatively, we can use Fibonacci heaps [see Chapter 2, section 5] to store nodes in $D(q)$ with keys $nsd(i)$'s. Since $nsd(i)$ are non-increasing, we use decrease-key operation to update $nsd(i)$'s when we scan forward star of the node selected. To select a node with minimum $nsd(i)$, we use find-min and delete-min operations. Since, in Fibonacci heaps, decrease-key and find-min takes $O(1)$ time, and delete-min takes $O((\log |D(q)|)$ time ($|D(q)|$ is the maximum number of elements in the heap at any stage of the algorithm), the time taken in Fibonacci heap implementation for step (ii) is $O\left(\sum_{i \in D(q)} (|F(i)| + |D(q)| \log |D(q)|)\right)$. We summarize this discussion in the following lemma.

Lemma 4.2: *The procedure reoptimization((p, q)) runs in* $O\left(\sum_{i \in D(q)} (|F(i)|+|B(i)|) + |D(q)|^2\right)$ *time in the array implementation and in* $O\left(\sum_{i \in D(q)} (|F(i)|+|B(i)|)+|D(q)|\log(|D(q)|)\right)$ *time in the Fibonacci heap implementation.*

The Main Algorithm

Having developed the required subroutines already, we can describe the algorithm for computing non-singleton shortest path between all pair of nodes.

We assume that right and left marginal costs of each arc are given or can be computed in $O(1)$ time. The algorithm has following steps : (i) find an optimal potential π, if it is not available; (ii) for each node $p \in N$, find non-singleton shortest distances from p to all other nodes using arc costs c_{ij}^{π}; (iii) using the formula,

$$\tilde{d}_i(j) = \tilde{d}_i^{\pi}(j) + \pi(i) - \pi(j)$$

calculate non-singleton shortest distances between all pair of nodes.

The step (ii) consists of following steps for each node p: (iia) find a shortest path tree T_p from node p to all other nodes (using Dijkstra's algorithm); (iib) build tree indices, thread and depth, for the tree T_p; (iic) for each $q \in SUCC(p)$, use procedure reoptimization (p, q), to find non-singleton shortest distance $\tilde{d}^{\pi}_{p}(q)$ from p to node q. Recall that, for nodes not in $SUCC(p)$, the unique tree path in T_p is not only the shortest but also non-singleton.

The steps of the algorithm are self explanatory, except for reoptimization$((p, q))$ (in step (iic)) for which we have already given the justification. Step (i) (finding an optimal potential) requires $O(nm)$ time. Step (iii) requires $O(n^2)$ time.

We now analyze the complexity of step (ii) for one node, say p; multiplying this complexity by n gives the complexity of step (ii).

Using Dijkstra's algorithm, finding a shortest path tree T_p (step (iia)) requires $O(n^2)$ time in array implementation, and $O(m+n \log n)$ time in Fibonacci heap implementation (see, for example, Ahuja et al. [1993]). Building tree indices for T_p (step (iib)) requires $O(n)$ time. In step (iic), we apply procedure reoptimization$((p, q))$ for each node $q \in SUCC(p)$.

By Lemma 4.2, procedure reoptimization$((p, q))$ requires $O\left(\sum_{i \in D(q)} (|B(i)|+|F(i)|) + |D(q)|^2\right)$ time in array implementation, and $O\left(\sum_{i \in D(q)} (|B(i)| + |F(i)| + |D(q)| \log |D(q)|)\right)$ time in

Fibonacci heap implementation. Note that $\bigcup_{q \in SUCC(p)} D(q) = N - \{p\}$. Thus summing the expression $\sum_{i \in D(q)} (|B(i)| + |F(i)|)$ over all q in $SUCC(p)$, we get the bound $\sum_{i \in N} (|B(i)| + |F(i)|) = m + m = O(m)$. Similarly the summation of $|D(q)|^2$ over all q in $SUCC(p)$ gives $O(n^2)$ bound. (Here we use the property that if $a_1, a_2, \ldots, a_k$ are positive integers, then $(a_1 + a_2 + \ldots + a_k)^2 > \sum_{i=1}^{k} a_k^2$ and the fact $\sum_{q \in SUCC(p)} |D(q)| < n$.) Summation of the term $|D(q)|$ log $(|D(q)|)$, since log $|D(q)| <$ log n, gives the bound $O(n \log n)$. Thus, summing respective complexity bounds for procedure reoptimization(p, q) given in Lemma 4.2, we get the complexity of step (iic) for a node p as $O(m+n^2)$ and $O(m+n \log n)$ in array and Fibonacci heap implementations, respectively. Since the complexity of this step dominates the complexity of all the other steps, the time complexity of step (ii) for node p is $O(n^2)$ in array implementation and $O(m+n \log n)$ in Fibonacci heap implementation. Hence, step (ii) (over all nodes) requires $O(n^3)$ in array implementation and $O(nm + n^2 \log n)$ in Fibonacci heap implementation. Since this step dominates the bound for step (i) and step (iii), the algorithm runs in $O(n^3)$ time in array implementation, and $O(nm + n \log n)$ time in Fibonacci heap implementation.

We summarize the discussion in the following theorem.

Theorem 4.4 : *The algorithm given in this section finds non-singleton shortest distance in $G(\bar{x})$ between all pairs of nodes in $O(n^3)$ time in array implementation, and in $O(nm + n^2 \log n)$ time in Fibonacci heap implementation.*

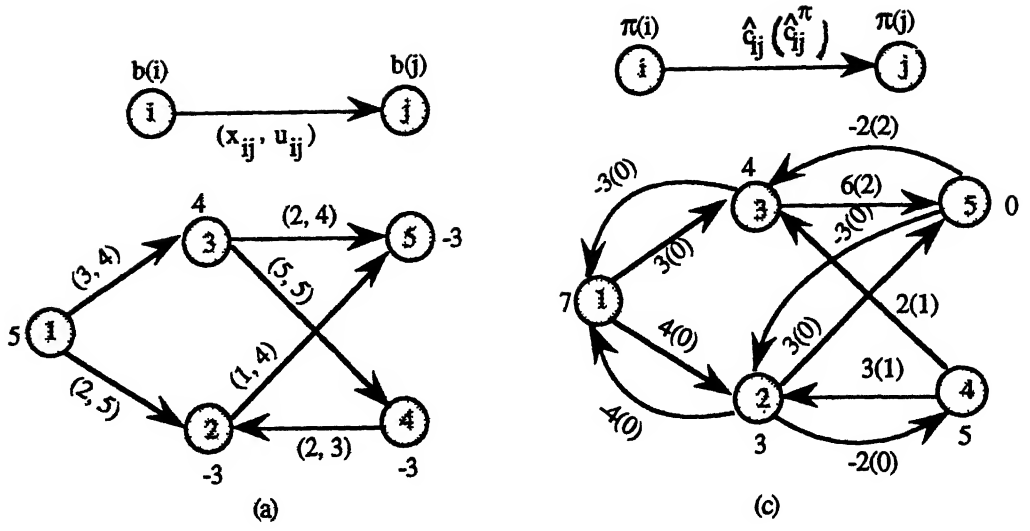
Apart from the storage requirement for the initial data, for every node p, we need an additional n-dimensional array for storing each of the following data : distance labels $d^{\pi}_{p}(i)$, $\check{d}_{p}(i)$, three indices pred(i), depth(i), thread(i). Thus, over all nodes we require $O(n^2)$ additional storage.

Theorem 4.4 together with Theorem 4.3 implies that one can find tolerance intervals of all arcs in a network for both minimum cost flow (MCF) and separable convex cost flow (SCCF) problems in $O(n^3)$ time in array implementation, and in $O(nm + n^2 \log n)$ time in Fibonacci heap implementation.

Example

Consider the example network in Figure 4.3(a) with the given flow shown on the arcs; the (convex) cost functions associated with arcs in Figure 4.3(a) are tabulated in Figure 4.3(b). The given flow is an optimal flow for SCCF problem defined on Figure 4.3(a) for the cost functions given in Figure 4.3(b). The residual network $G(\bar{x})$ for the flow $\bar{x}$ is given in 4.3(c); a set of optimal node potentials are shown alongside of nodes.

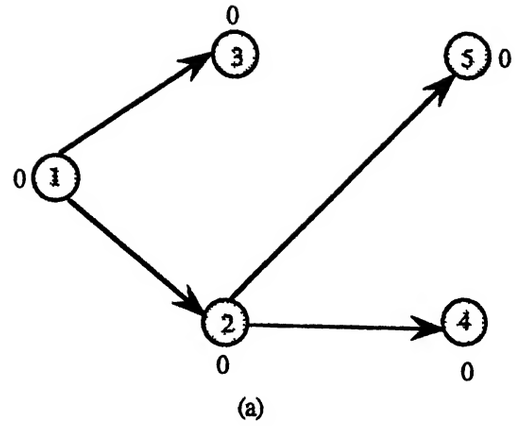
Figure 4.4 illustrates the computation of the non-singleton shortest path distance $\check{d}^{\pi}_{1}(2)$ in $G(\bar{x})$ from node 1 to node 2. Figure 4.4(a) shows a shortest path tree T_1 from node 1. Node 2 is in SUCC(1) (that is, a successor of node 1). Therefore, we apply the reoptimization procedure. We first delete the arc (1, 2). Then we apply the procedure setup-potentials to initialize distance labels for node in $D(2)$; these labels denote the shortest distances to nodes in $D(2)$ passing only through nodes in N-$D(2)$ using arc lengths $\hat{c}^{\pi}_{ij}$. Figure 4.4(b) shows these labels. Then we



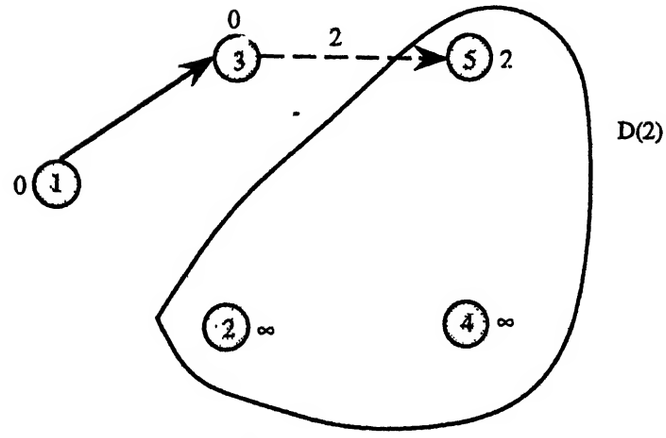
(i, j)	$C_{ij}(x_{ij})$	x_{ij}	$C^{-}_{ij}(x_{ij})$	$C^{+}_{ij}(x_{ij})$
(1, 2)	x^2_{12}	2	4	4
(1, 3)	$1/2\ x^2_{13}$	3	3	3
(2, 5)	$1/2\ x^2_{25}+2x_{25}$	1	3	3
(3, 4)	$-2\ x_{34}$	5	-2	-2
(3, 5)	$\max\{2x_{35}, 6\ x_{35}-8\}$	2	2	6
(4, 2)	$\max\{2x_{42}, 3\ x_{42}-2\}$	2	2	3

(b)

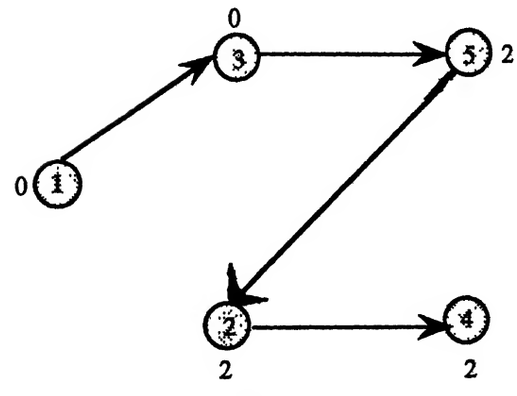
Figure 3 (a) An example network with an optimal flow
(b) (convex) cost functions associated with each arc
(c) The residual network with optimal node potentials.



(a)



(b)



(c)

Figure 4.4 (a) Shortest path tree T_j from node 1 with initial shortest distance with respect to $\hat{c}_{ij}^{\pi}$

(b) Shortest distance labels in G(x) - (1, 2) (nsd's) after the procedure set-up poentials.
(c) a nonsingleton shortest path from node 1 to node 2.

apply the Dijkstra's algorithm with N-$D(2)$ as permanently labeled nodes. The figure 4.4(c) shows a resultant non-singleton shortest path from node 1 to node 2. The non-singleton shortest distance $\tilde{d}_1^{\pi}(2)$ (with respect to arc length $\hat{c}_{ij}^{\pi}$) is 2. By Equation 4.25, the son-singleton shortest distance $\tilde{d}_1(2) = \tilde{d}_1^{\pi}(2) + \pi(1) - \pi(2) = 2 + 7 - 3 = 6$.

By applying similar procedure, we find $\tilde{d}_2(1) = 1$, and hence $\tilde{d}_2(1) = 1 + 3 - 7 = -3$. Therefore, by Theorem 4.3, the tolerance interval of arc (1, 2) is $[\alpha_{kl}, \beta_{kl}] = [-\tilde{d}_2(1), \tilde{d}_1(2)] = [3, 6]$. Hence any convex function $\tilde{C}_{12}(x_{12})$ of x_{12} with at least one of its marginal costs ($\tilde{C}_{12}^{-}(x_{12})$ and ($\tilde{C}_{12}^{+}(x_{12})$) in the interval is a valid perturbation. Note that when the function is of linear type, i.e., $c_{12}x_{12}$, it is a valid perturbation if c_{12} takes any value in [3, 6]; this is the cost sensitivity analysis in the traditional sense in the minimum cost flow problem.

Chapter 5

INVERSE LINEAR PROGRAMMING AND NETWORK FLOW PROBLEMS

5.1 INTRODUCTION

The well known, widely used linear programming problem is to minimize linear (cost) function over a given system of linear constraints (polyhedron) on R^m. The cost objective function is represented by the vector product $cx = \sum_{j=1}^{m} c_j x_j$, where $c \in R^m$ and $x \in R^m$. The vector c is referred to as the cost (vector), c_j as the cost coefficient of the variable x_j. On the other hand, suppose c is a priori estimated cost and $\bar{x}$ is a feasible solution of the given system of linear constraints, say P. Let t denote any arbitrary cost for which $\bar{x}$ is the minimum cost solution over P (that is, the cost function tx attains its minimum at $\bar{x}$ over P). We refer to such a cost (vector) as conormal to P at $\bar{x}$. Given $\bar{x} \in P$, we call the problem of finding a conormal (to P at $\bar{x}$), whose 'variance' from the priori estimated cost c is minimum as the ***inverse linear programming problem (ILP)***. Note that we can take $t = 0$ as the solution to the above problem when there is no 'variance' objective, which has no meaning. When the given system of linear constrains is a system of flow conservation equations and flow bound constraints defined on a directed graph, we call the corresponding ILP as the ***inverse network flow problem (IF)***.

One can interpret the given solution $\bar{x} \in P$ as the observed solution of the system P.

We measure variation of a vector t from the cost c by a real-valued function $k(t-c)$, satisfying the following properties :

(i) $k(0) = 0$ and $t \neq 0 \iff k(t) > 0$,

(ii) $k(t_1 + t_2) \leq k(t_1) + k(t_2)$,

(iii) $k(\lambda t) = \lambda\, k(t)$ for $\lambda > 0$.

We refer to a function k with above properties as an ***unorm*** and $k(t-c)$ as the ***k-variance***. Thus an ILP is minimizing a k-variance from the priori estimated cost c over all conormals to P at $\bar{x}$. Since the conormals to P at $\bar{x}$ are the cost for which $\bar{x}$ is optimal, ILP is about perturbing a priori estimated cost to a minimum k-variance conormal for which an observed solution of the given system becomes a minimum cost solution. The difference between an LP and ILP is that while LP determines a minimum cost solution of a linear system of constraints, ILP infers a conormal to the given solution of a linear system of constraints with minimum 'variance' from the given cost.

Burton and Toint [1992] studied inverse (multiple source) shortest paths problems; the problem is to find minimum least square variance nonnegative arc costs from a priori estimated cost such that the given set of (observed) paths become shortest paths with respect to the newly determined cost. They later investigated (multiple source) shortest paths problems in which the cost coefficients of arcs in given disjoint subsets are linearly correlated (Burton and Toint [1994]). Here we point out that inverse (multiple source) path problem without any

constraints on arc costs is a special case of ILP and inverse (single source) shortest paths problem without any constraints on arc costs is a special case of IF. Burton and Toint's work is mainly focused on developing algorithms for inverse shortest path problems specializing Goldfarb and Idnani's [1983] dual algorithm for strictly convex quadratic programs. Their motivation for studying inverse shortest paths problems is due to its applications both in transportation research and computerized tomography. In the first case, the question was to recover travel costs as perceived by network users from the knowledge of their actual route choices. The second example aimed at reconstructing the wave speeds in a discretized medium based on their paths in this medium.

In this chapter, we study the duality of inverse network flow problems. Mainly, we show that duals of inverse network flow problems with respect to standard variances such as Euclidean, weighted rectilinear and weighted maximum variances are standard network flow problems. We also characterize solutions and the optimal values whenever possible.

For this purpose, we first develop the duality theory for inverse linear problems from duality results in convex analysis. We mostly adopt results from the book "Convex Analysis" by Rockafellar [1970]. Applying this theory for inverse network flow problems, we show the following results:

(i) The optimal solutions to Euclidean inverse network flow problem (EIF) and its dual give a decomposition of the given

vector c; the dual is equivalent to a quadratic (strictly convex) cost flow problem.

(ii) The dual of weighted rectilinear network flow problem (WRIF) is a minimum cost flow problem. For the rectilinear network flow problem, a minimum cost arc-disjoint augmenting cycles corresponds to an optimal solution of the dual.

(iii) The weighted maximum inverse network flow problem and the maximum inverse network flow problem are equivalent to solving minimum cost-to-weight ratio augmenting cycle problem and minimum mean augmenting cycle problem, respectively.

(iv) The dual of weighted rectilinear inverse spanning tree problem is a transportation problem and the dual of rectilinear inverse spanning tree problem is an assignment problem.

In Section 5.2, we review definitions and certain results from convex analysis. We develop the duality theory for inverse linear programming in Section 5.3. We consider inverse network flow problems in Section 5.4. In Section 5.5, we study inverse spanning tree problem.

5.2 PRELIMINARIES

A subset S of R^m is *convex* if $(1-\lambda)x + \lambda y \in S$ whenever $x \in S$. $y \in S$ and $0 < \lambda < 1$. A convex set K is a *convex cone* if $\lambda x \in K$ whenever $x \in K$ and $\lambda > 0$. The *convex cone $K(S)$ generated by a convex set S* is defined as the set :

$$K(S) = \{ \lambda x | x \in S, \lambda \geq 0 \}$$

Its *translate*

$$S+a = \{ x+a \mid x \in S \}, \; a \in R^m,$$

and the *scalar multiple*

$$\lambda S = \{ \lambda x \mid x \in S \}, \; \lambda > 0,$$

of a convex set S are convex. The *symmetric reflection* of S across the origin is $-S = (-1)S$. A convex set is said to be *symmetric* if $S = -S$.

A subset M of R^m is called an *affine set* if $(1-\lambda)x + \lambda y \in M$ whenever $x \in M$, $y \in M$ and $\lambda \in R$. The *affine hull* of a set S, denoted by aff S, is the set of all vectors of the form $\lambda_1 x_1 + \ldots + \lambda_k x_k$, such that $x_i \in S$, $i = 1, 2, \ldots, k$, and $\lambda_1 + \ldots + \lambda_k = 1$.

The *Euclidean norm* of $x \in R^m$ is, by definition,

$$|| x || = \left(\sum_{j=1}^{m} | x_j |^2 \right)^{1/2}.$$

We shall denote by B the *Euclidean unit ball* in R^m, that is,

$$B = \{ x \mid || x || \leq 1 \}.$$

A set S in R^m is *bounded* if $S \subset \lambda B$ for some $\lambda \in R$.

For any set S in R^m, the *closure* cl S and *interior* int S can be defined as the following sets :

$$\text{cl } S = \{ x \mid (x+\varepsilon B) \cap S \neq \phi \text{ for every } \varepsilon > 0 \},$$
$$\text{int } S = \{ x \mid \text{for some } \varepsilon > 0, \; x + \varepsilon B \subset S \}.$$

The *relative interior* ri S of a convex set S is the interior of S when S is regarded as a subset of its affine hull aff S. Thus,

the set ri S of S is defined as the set :

$$\text{ri } S = \{ x \in \text{aff } S | \text{ for some } \epsilon > 0, (x+\epsilon B) \cap (\text{aff } S) \subset S \}$$

A convex set C is said to be ***relatively open*** if ri $C = C$, and ***closed*** if cl $C = C$. Note that int S can be empty even when ri S is non-empty. When int $S \neq \phi$, int S = ri S. We call y as an interior points of S if $y \in$ int S.

Let f be a function whose values are real or $\pm \infty$ and whose domain is a subset S of R^n. The function f is convex if

$$f((1-\lambda)x + \lambda y) < (1-\lambda)\alpha + \lambda\beta, \quad 0 < \lambda < 1,$$

whenever $f(x) < \alpha$ and $f(y) < \beta$. Suppose f takes values only in $(-\infty, +\infty)$. Then f is convex on a set S in R^m if

$$f((1-\lambda)x + \lambda y) \leq (1-\lambda)\, f(x) + \lambda f(y), \; 0 < \lambda < 1, \qquad (5.1)$$

whenever $x \in S$ and $y \in S$. The ***effective domain*** of a convex function f on S, denoted by dom f, is the set :

$$\text{dom } f = \{ x | \, f(x) < \infty \}.$$

A convex function f on R^m is said to be ***proper*** if its values are real or $+\infty$ and $f(x)$ is finite for at least one x. A convex function f is said to be ***closed*** if $\{x | f(x) \leq \alpha\}$ is closed for every $\alpha \in R$.

The ***conjugate*** $f^*(y)$ of a convex function $f(x)$ in R^m is defined by

$$f^*(y) = \sup_{x \in R^m} \left\{ xy - f(x) \right\}. \qquad (5.2)$$

Let S be any non-empty convex set. The ***polar*** S° of S is defined by

$$S^\circ = \{ y | xy \leq 1 \text{ for every } x \in S \}. \qquad (5.3)$$

When K is a non-empty convex cone, the definition of polar K° of K reduces to the one given below :

$$K^{\circ} = \{ y | \; xy \leq 0 \quad \text{for every } x \in K \}. \tag{5.4}$$

We associate the following convex functions to a convex set S. The *indicator function* $\delta(x|S)$ of S is defined by

$$\delta(x|S) = \begin{cases} 0 & \text{if } x \in S, \\ +\infty & \text{if } x \notin S. \end{cases} \tag{5.5}$$

The *support function* $\delta^{*}(y|S)$ of S is defined by

$$\delta^{*}(x|S) = \sup\{ xy | \; y \in S \}. \tag{5.6}$$

The *gauge function* $\gamma(x|S)$ of S is defined by

$$\gamma(x|S) = \inf\{ \lambda \geq 0 | \; x \in \lambda S \}, \; S \neq \phi. \tag{5.7}$$

We call a real valued function $k(t)$ as an unorm if it satisfies the following properties :

(i) $k(0) = 0$, and $k(t) > 0$ for every $t \neq 0$. (5.8)

(ii) $k(\lambda t) = \lambda k(t)$ for every $\lambda > 0$, every t. (5.9)

(iii) $k(t_1+t_2) \leq k(t_1) + k(t_2)$ for every t_1, t_2. (5.10)

An unorm $k(t)$ is obviously a convex function. When an unorm $k(t)$ satisfies $k(t) = k(-t)$ for every t, then it becomes a norm.

The polar $k^{\circ}(y)$ of an unorm $k(t)$ is defined by

$$k^{\circ}(y) = \inf \{\mu \geq 0 | \; yt \leq \mu k(t) \text{ for every } t\}. \tag{5.11}$$

The above formula can also be written as

$$k^{\circ}(y) = \sup_{t \neq 0} \frac{yt}{k(t)}. \tag{5.12}$$

In the following proposition, we collect some useful results from Rockafellar [1970] for future reference; we refer to the corresponding result in the above reference along with each result.

Preposition 5.1 : *Let k be an unorm and k^o be its polar. Let*

$$R = \{t | k(t) \leq 1\}, \tag{5.13}$$

and $$R^o = \{y | k^o(y) \leq 1\}. \tag{5.14}$$

Then,

(a) $k(t) = \gamma(t|R)$; $k^o(y) = \gamma(y|R^o)$. *The set R and R^o are uniquely defined closed bounded convex sets containing the origin as its interior point, and they are polar to each other (Theorem 15.1, Theorem 14.5, Corollary 14.5.1).*

(b) $k(t) = \delta^*(t|R^o)$; $k^o(y) = \delta^*(y|R)$ *(Theorem 14.5)*

(c) $k^*(y) = \delta(y|R^o)$ *(Theorem 13.2).*

A set P of vectors in R^m is called a (convex) ***polyhedron*** if

$$P = \{ x | Ax \leq b \},$$

where A is a nxm matrix and $b \in R^n$. When $b = 0$, the polyhedron P is a polyhedral cone.

5.3 INVERSE LINEAR PROGRAMMING PROBLEM

Let $\bar{x}$ be the observed solution to a system P of linear constraints; that is, let $\bar{x}$ be a point in a polyhedron P. Let k be an unorm, and c be the priori estimated cost vector in R^m. The inverse linear programming problem (ILP) is to find a vector $\bar{t}$ such that k-variance $k(t-c)$ attains its minimum at $\bar{t}$ over the set

of all vector t with the property that the linear function tx attains its minimum over P at $\bar{x}$.

A *conormal* to the polyhedron P at $\bar{x}$ is a vector t which makes an acute angle with every line segment in P at $\bar{x}$, i.e., $t(x-\bar{x}) \geq 0$ for every $x \in P$. By definition, t is a conormal to P at $\bar{x}$ if and only if $tx \geq t\bar{x}$ for every $x \in P$; that is, the linear function tx attains its minimum at $\bar{x}$. The set of all conormal vectors to P at $\bar{x}$ is called the *conormal cone* to P at $\bar{x}$, and is denoted by $T(\bar{x},P)$. The symmetric reflection of $T(\bar{x}, P)$ across the origin is $N(\bar{x}, P) = -T(\bar{x}, P)$; it is known as the normal cone. A vector in $N(\bar{x}, P)$ is a normal to P at $\bar{x}$. With this definition, we can formulate ILP as minimizing k-variance from c, $k(t-c)$, over all conormals of P at $\bar{x}$, that is, over the conormal cone $T(\bar{x}, P)$. Thus ILP is:

$$\text{Minimize } f(t) = k(t-c) \tag{5.15a}$$

such that

$$t \in T(\bar{x}, P). \tag{5.15b}$$

The optimal solution to ILP is called a *minimal k-variance conormal* to P at $\bar{x}$ from c.

Feasibility Cone to P at $\bar{x}$

A *feasible direction* y to P at $\bar{x}$ is by definition a vector $y \in \bar{x} + \theta y \in P$ for some $\theta > 0$, i.e.,

$$\bar{x} + \theta y = x \text{ for some } x \in P, \ \theta > 0,$$

or, $$y = \lambda(x - \bar{x}), \quad \lambda \geq 0 \text{ and some } x \in P.$$

0 vector is also considered a feasible direction at any point $\bar{x}$ to P. Thus the set of all feasible direction to P at $\bar{x}$ is the cone $K(P-\bar{x})$ generated by the convex set $P - \bar{x}$. We call $K(P-\bar{x})$ as the

feasible cone to P at $\bar{x}$. Since $P - \bar{x}$ is a polyhedron containing the origin, $K(P-\bar{x})$ is a polyhedral cone (see Rockafellar [1970, Corollary 19.7.1]). We state this result as the following lemma.

Lemma 5.1 : *The feasible cone $K(P - \bar{x})$ to P at $\bar{x}$ is a polyhedron.*

Now we study the polarity relationship between the convex cones $T(\bar{x}, P)$ and $K(P - \bar{x})$ before considering the existence of minimal k-variance conormal to P at $\bar{x}$ from c.

We define the copolar K^* of a convex cone K as the set $\{y \mid yx \geq 0$ for every $x \in K\}$. Recall that the polar K° of K, by definition, is the set $\{y \mid yx \leq 0$ for every $x \in K\}$. For a non-empty convex cone K, $K^* = -K^\circ = (-K)^\circ$. The following results are straightforward conversion of polarity results on polyhedral cones (see Schrijver [1986, Theorem 9.1]) to copolarity results.

Lemma 5.2 : *Let K be a non-empty convex polyhedral cone. Then:*

(a) K^* is a non-empty polyhedral cone.

(b) $K^{**} = K$.

(c) If $K = \text{cone}\{x_1, \ldots, x_t\}$, then $K^* = \{y \in R^m \mid y\, x_j \geq 0$ for $j = 1, \ldots, t$, and converse also holds, i.e., given K^*, K is the cone generated by $\{x_1, \ldots, x_t\}$.

The following theorem gives the copolarity between the conormal cone $T(\bar{x}, P)$ and the feasibility cone $K(P-\bar{x})$ relative to a point $\bar{x}$ in the polyhedron P.

Theorem 5.1 : *Let $\bar{x}$ be a point in a polyhedron P.*

(a) The conormal cone $T(\bar{x}, P)$ and the feasibility cone $K(P - \bar{x})$ are copolar to each other.

(b) Both $T(\bar{x}, P)$ and $K(P - \bar{x})$ are (non-empty) polyhedral cones.

(c) $-K(P-\bar{x})$ is the polar of $T(\bar{x}, P)$.

Proof : By definition, $t \in T(\bar{x}, P)$ if $t(x - \bar{x}) \geq 0$ for every $x \in P$, i.e., $ty \geq 0$ for every $y \in P - \bar{x}$. .Also, if $ty \geq 0$ for some y, then $t(\lambda y) \geq 0$ for every $\lambda \geq 0$. Hence $t \in T(\bar{x}, P)$ if and only if $t(\lambda y) \geq 0$ for every $y \in P - \bar{x}$ and for every $\lambda \geq 0$, i.e., $ty \geq 0$ for every $y \in K(P - \bar{x})$. Hence $T(\bar{x}, P) = K^*(P - \bar{x})$.

From Lemma 5.2(b),

$$T^*(\bar{x}, P) = (K(P - \bar{x}))^{**} = K(P - \bar{x}).$$

By Lemma 5.1, $K(P - \bar{x})$ is a non-empty polyhedral cone. It follows from Lemma 5.2(a) that $T(\bar{x}, P) = [K(P - \bar{x})]^*$ is also a non-empty polyhedral cone.

(c) follows from the definitions of polar and copolar. ■

Now we turn our attention to the existence and finiteness of the optimum solutions to the inverse linear programming problem (*ILP*). We make use of the following Lemma which is a partial statement of the results in Rockafellar [1970, Theorem 27.1(d) and 27.3].

Lemma 5.3 : *Let h be a closed proper convex function on R^m which attains its minimum on a non-empty bounded set, and let S be a non-empty closed convex set. Then h attains its infimum over S.*

Now we state and prove the following theorem.

Theorem 5.2 : *A minimal k-variance conormal to P at $\bar{x}$ from c (an optimal solution to ILP) exists and its value is finite, that is, the inverse linear programming problem has finite optimal solutions.*

Proof : The objective function in *ILP* is $f(t) = k(t - c)$, where k is an unorm. Thus, by definition, $f(t)$ attains infimum, which is zero, at the unique point c. Furthermore, the set of feasible solutions $T(\bar{x}, P)$ is non-empty polyhedral cone by Theorem 5.1(b), and hence a non-empty closed convex set.

By applying Lemma 5.3 with $h(t) = f(t) = k(t-c)$ and $S = T(\bar{x}, P)$, we conclude that $k(t-c)$ attains its infimum over $T(\bar{x}, P)$, i.e., optimal solution to *ILP* exists. The finiteness of the optimal solution follows from the fact that $k(t-c)$ is finite everywhere on R^m. ■

Duality in Inverse Linear Programming

We study the duality in inverse linear programming. As defined in the previous section, let $c \in R^m$, $x \in P$, P a polyhedron in R^m, and k be an unorm on R^m. Let k^o be the polar of k.

The inverse linear programming problem (*ILP*) is the problem of finding a minimal k-variance conormal to P at $\bar{x}$ from c, which is formulated as below:

Minimize $k(t - c)$

subject to

$t \in T(\bar{x}, P)$,

where $T(\bar{x}, P)$ is the conormal cone to P at $\bar{x}$.

Now we proceed formally to define the dual DILP of ILP. We call a feasible direction y to P at $\bar{x}$ as *k^o-normalized* if $k^o(y) = 1$. A direction y is called *c-descent* direction if $cy < 0$; its value is $-cy$. We define the dual DILP of ILP as the problem of either finding a ***maximal c-descent k^o-normalized feasible direction*** y to

P at $\bar{x}$ if one exist or concluding there is no c-descent (k^o-normalized) feasible direction. In the latter case the null vector is the optimum solution and the optimum value is zero. We call the optimum value of the DILP as the maximum c-descent over all k^o-normalized feasible directions to P at $\bar{x}$ (and the null vector).

Theorem 5.3 : *DILP is :*

$$\textit{Max } -cy \tag{5.16a}$$

subject to

$$y \in K(P-\bar{x}) \cap R^o, \tag{5.16b}$$

where $R^o = \{y | k^o(y) \leq 1\}$, *and* k^o *is the polar of* k.

Proof : The feasible set $K(P-\bar{x}) \cap R^o$ contains all feasible directions to P at $\bar{x}$ with $k^o(y) \leq 1$. Therefore, an optimum solution to (5.16) is a maximal c-descent feasible direction over all feasible directions with $k^o(y) \leq 1$.

Note that $K(P-\bar{x}) \cap R^o$ contains zero direction (Theorem 5.1(b) and Proposition 5.1). Therefore, if there is no c-descent feasible direction to P at $\bar{x}$, 0 is obviously an optimum solution.

Suppose there is a c-descent feasible direction and $\bar{y}$ is an optimal solution to (5.1b). The proof is complete if $k^o(\bar{y}) = 1$, since there is c-descent direction, $\bar{y} \neq 0$ and $k^o(\bar{y}) > 0$. Suppose $k^o(\bar{y}) < 1$. Let $\hat{y} = \bar{y}/k^o(\bar{y})$; the $k^o(\hat{y}) = 1$. Therefore $\hat{y} \in K(P-\bar{x}) \cap R^o$, since $K(P-\bar{x})$ is a cone. Further $-c(\hat{y}) = 1/k^o(\bar{y})\ (-c\bar{y}) > -c\bar{y}$, contradicting $\bar{y}$ is an optimal solution. Therefore, $k^o(\bar{y}) = 1$. ■

Theorem 5.4 : *A maximal c-descent k^O-normalized feasible direction to P at $\bar{x}$ (an optimal solution to DILP) exists and its value is finite. Moreover, there exists an extreme point of $K(P-\bar{x})\cap R^O$ which is a maximal c-descent k^O-normalized feasible direction to P at $\bar{x}$.*

Proof : $K(P-\bar{x})$ is a polyhedron cone (Theorem 5.1(b)) and R^O is a closed bounded convex set having the origin as its interior point (Proposition 5.1). Therefore $K(P-\bar{x})\cap R^O$ is a closed bounded convex set containing the origin. The supremum of the linear function $-cy$ over a nonempty closed bounded convex set is finite, and it is attained at some extreme point of the set (Rockafellar [1970, Corollary 32.3.2]). Hence the theorem. ■

The following is the weak duality result in inverse linear programming.

Theorem 5.5 (Weak Duality) : *For any $t \in T(\bar{x}, P)$ and $y \in K(P-\bar{x})\cap R^O$, $(-c)y \le k(t-c)$.*

Proof : Consider arbitrary $t \in T(\bar{x}, P)$ and $y \in K(P-\bar{x})\cap R^O$, where $R = \{t|k(t) \le 1\}$ and $R^O = \{y|k^O(y) \le 1\}$. Then $k(t-c) = \gamma(t-c|R) = \text{Inf}\{\mu \ge 0|(t-c) \in \mu R\}$. If $k(t-c) = 0$, since k is an unorm, $(t-c) = 0$. Therefore $(t-c)y = 0 = k(t-c)$. Suppose $k(t-c) = \bar{\mu} > 0$. Then $(t-c) \in \bar{\mu} R$, i.e., $(t-c)/\bar{\mu} \in R$. Since $y \in R^O$, and R and R^O are (non-empty) closed convex sets polar to each other, (Preposition 5.1) $((t-c)/\bar{\mu}).y \le 1$. Therefore, $(t-c).y \le \bar{\mu} = k(t-c)$. Thus, $(t-c)y \le k(t-c)$.

We have $ty \ge 0$, since $t \in T(\bar{x}, P)$ and $y \in K(P - \bar{x})$, and $T(\bar{x}, P)$ and $K(P - \bar{x})$ are (non-empty) closed convex cones copolar to each other (Lemma 5.1). Consequently, $(-c)y \le (t-c)y \le k(t-c)$. ■

We know that, by Theorem 5.2 and Theorem 5.4, both the inverse linear programming problem (*ILP*) and its dual (*DILP*) have finite optimal solutions. Further, from the above weak duality theorem, we have

$$\text{Max } \{(-c)y \mid y \in K(P - \bar{x}) \cap R^{O}\}$$
$$\leq \text{Min}\{f(t) = k(t-c) \mid t \in T(\bar{x}, P)\}. \qquad (5.17)$$

Now we prove the duality result for inverse linear programming; that is, max = min in the relation (5.17). We first prove the following result.

Lemma 5.3 : *The conjugate $f^{*}(y)$ of the objective function $f(t) = k(t-c)$ in the inverse linear programming problem (ILP) is given by*

$$f^{*}(y) = cy + \delta(y \mid R^{O})$$

Proof : We have $f^{*}(y) = cy + k^{*}(y)$ Rockafellar [1970, p. 140]. By Proposition 5.1, $k^{*}(y) = \delta(y \mid R^{O})$. ■

The dual (*DILP*) is equivalent to the problem: Maximize $\{-[cy + \delta(y \mid R^{O})] \mid y \in K(P - \bar{x})\}$ [Here we use the convention that the objective function value of an infeasible solution is $-\infty$ in a maximization problem]. By Lemma 5.3, this can again be rewritten as the following :

$$\text{Maximize } \{-f^{*}(y) \mid y \in K(P - \bar{x})\},$$

where $f(t) = k(t-c)$.

Now we state and prove the duality theorem.

Theorem 5.6 (Duality theorem for inverse linear programming): *Let $c \in R^{m}$ and $x \in P$, where P is a polyhedron in R^{m} and let k be an unorm and k^{O} be its polar. Then the values of minimal k-variance*

conormal to P at $\bar{x}$ *from c and maximal c-descent* k^O*-normalized feasible direction to P at* $\bar{x}$ *are equal; that is,*

$$\begin{aligned} &\textit{Max } \{-cy \mid y \in K(P - \bar{x}) \cap R^O\} \\ &= \textit{Min } \{k(t-c) \mid t \in T(\bar{x}, P)\}, \end{aligned} \tag{5.18}$$

where $R^O = \{y \mid k^O(y) \leq 1\}$.

Proof : By the remarks preceding the theorem, we can rewrite (5.18) as follows :

$$\text{Max } \{-f^*(y) \mid y \in K(P - \bar{x})\} = \text{Min } \{f(t) \mid t \in (\bar{x}, P)\}, \tag{5.19}$$

where $f^*(y) = cy + \delta(y \mid R^O)$ and $f(t) = k(t-c)$, which has to be proved.

We now make use of a result which is a direct corollary of Theorem 31.4, Rockafellar [1970], which is the following:

Let h be a real-valued convex function and K a non-empty polyhedral cone in R^m and K^* the copolar of K. Then

$$\text{Inf}\{h(t) \mid t \in K\} = \sup \{-h^*(y) \mid y \in K^*\}. \tag{5.20}$$

By Theorem 5.1, $T^*(\bar{x}, P) = K(P - \bar{x})$ and $T(\bar{x}, P)$ is a non-empty polyhedral cone. Further, since $k(t)$ is real-valued, so is $f(t) = k(t-c)$. Now, by letting $K = T(\bar{x}, P)$ and $h(t) = f(t)$ in (5.20) we have

$$\text{Inf}\{f(t) \mid t \in T(\bar{x}, P)\} = \sup\{-f^*(y) \mid y \in K(P - \bar{x})\} \tag{5.21}$$

Since, by Theorem 5.1 and Theorem 5.4, both ***ILP*** and its dual ***DILP*** have finite optimal solutions, we can replace Inf and sup in (5.21) by min and max respectively. Thus we have proved (5.19). ■

Necessary and Sufficient Conditions for Optimality in Inverse Linear Programming :

The following result directly follows from the duality theorem and the weak duality theorem.

Theorem 5.7 : *Let $\bar{t}$ be a feasible solution to ILP and $\bar{y}$ a feasible solution to its dual DILP. Then $\bar{t}$ and $\bar{y}$ are optimal solutions to ILP and DILP respectively if and only if*

$$k(\bar{t} - c) = -c\bar{y}. \tag{5.22}$$

Proof : The necessity follows from the duality (Theorem 5.6). The sufficiency by the weak duality (Theorem 5.5). ■

We derive Kuhn-Tucker conditions for *ILP* in Theorem 5.8. One can also derive these conditions from Theorem 31.4, Rockafellar [1970].

Theorem 5.8 : *Let $\bar{t}$ be a feasible solution to ILP and $\bar{y}$ a feasible solution to its dual DILP. Then, $\bar{t}$ and $\bar{y}$ are optimal solutions to ILP and DILP respectively if and only if the following conditions hold :*

(a) The linear function $(\bar{t}-c)y$ attains its maximum over R^o at $\bar{y}$, that is,

$$k(\bar{t}-c) = \delta^*((\bar{t}-c)|R^o) = (\bar{t}-c).\bar{y}. \tag{5.23a}$$

(b) $\bar{t}$ and $\bar{y}$ are perpendicular to each other, that is,

$$\bar{t}.\bar{y} = 0. \tag{5.23b}$$

Proof : We have, by Proposition 5.1, $k(t-c) = \delta^*(t-c|R^o)$. Therefore, since R^o is a non-empty closed bounded convex set,

$$k(\bar{t}-c) = \delta^*(\bar{t}-c|R^o) = (\bar{t} - c).\hat{y} \text{ for some } \hat{y} \in R^o.$$

Sufficiency : By (a), $(\bar{t}-c)\bar{y} = \delta^*(\bar{t}-c|R^o) = k(\bar{t}-c)$.

By (b), $(\bar{t}-c)\bar{y} = -c\bar{y}$. Therefore, $k(\bar{t}-c) = -c\bar{y}$.

Hence, by Theorem 5.7, we have the sufficiency.

Necessity : Since $\bar{t}$ and $\bar{y}$ are the optimal solutions to *ILP* and *DILP*, we have $k(t-c) = -c\bar{y}$. Since $\bar{t} \in T(\bar{x}, P)$ and $\bar{y} \in K(P - \bar{x})$, we have $\bar{t}.\bar{y} \geq 0$, and consequently $-c\bar{y} \leq (\bar{t}-c)\bar{y}$. From the remarks at the beginning, we have $k(\bar{t}-c) = (\bar{t}-c)\hat{y}$ for some $\hat{y} \in R^o$. Hence $(\bar{t}-c)\hat{y} = k(\bar{t}-c) = -c\bar{y} \leq (\bar{t}-c)\bar{y}$. Since $\bar{y} \in R^o$ and $k(\bar{t}-c) = \delta^*(\bar{t}-c|R^o)$ imply that $(\bar{t}-c)\bar{y} \leq (\bar{t}-c)\hat{y} = k(\bar{t}-c) = \delta^*(\bar{t}-c|R^o)$, which in turn implies (a). Further, $k(\bar{t}-c) = -c\bar{y} = (\bar{t}-c)\bar{y}$ implies $\bar{t}.\bar{y} = 0$ which proves (b). ■

The existence of finite optimal solutions to the *ILP* and *DILP* together with Theorem 8 give the following optimality conditions for *ILP* and *DILP*.

Theorem 5.9 : *A feasible solution $\bar{t}(\bar{y})$ of IP (DILP) is an optimal solution if and only if there exists a dual (primal) feasible solution $\bar{y}(\bar{t})$ satisfying the following conditions.*

(a) $\delta^*(\bar{t}-c|R^o) = (\bar{t}-c).\bar{y}$

(b) $\bar{t}.\bar{y} = 0$.

Note that, when $\bar{t}$ is a minimal k-variance conormal to P at $\bar{x}$ from c and $\bar{y}$ is a maximal c-descent k^o-normalized feasible direction to P at $\bar{x}$, we have

$$k(\bar{t}-c) = \delta^*(\bar{t}-c|R^o) = (\bar{t}-c)\bar{y} = -c\bar{y}. \qquad (5.24)$$

ILP on Systems of Linear Constraints in Standard Form

We have treated so far the polyhedron P and the conormal cone $T(\bar{x}, P)$ and the feasibility cone $K(P - \bar{x})$ as sets. Now we treat

the polyhedron P as a system of linear constraints. We now obtain the representations for $T(\bar{x}, P)$ and $K(P - \bar{x})$ as a system of linear constraints. These representations are important in solving the inverse linear programming problem.

We choose the following form for a system of linear constraints, denoted by $\bar{P}$:

$$\mathcal{A}x = b \tag{5.25}$$

$$l_j \leq x_j \leq u_j \text{ for } j = 1,2, \ldots, m \tag{5.26}$$

Here $\mathcal{A}$ is a nxm real matrix, b is nx1 real vector, x is a arbitrary point in R^m, u_j's and l_j's are real numbers. We assume u_j as ∞ when there is no upper bound, and l_j as $-\infty$ when there is no lower bound. The coordinates x_j of x are called variables. The vector b is called as the right-hand side vector, and u_j's as upper bounds and l_j's as lower bounds. The above representation is usually known as the ***standard form*** in linear programming. The other representations can be easily transformed to the above form.

Let $\bar{x}$ be a solution to $\bar{P}$. As we shall see in the following discussion, the vector b does not play any role in the representation of $T(\bar{x}, \bar{P})$ and $K(\bar{P} - \bar{x})$. The role of the upper bounds and lower bounds is limited to deciding the ***boundary*** status of the variables, that is, deciding whether the variable is at its lower/upper bound.

We define index sets **U**, **V** and **W** for a given solution $\bar{x}$ in P as below :

$$\mathbf{U} = \{j \mid l_j < \bar{x}_j < u_j\}, \tag{5.27a}$$

$$\mathbf{V} = \{j \mid \bar{x}_j = l_j\}, \tag{5.27b}$$

$$\mathbf{W} = \{j \mid \bar{x}_j = u_j\}. \tag{5.27c}$$

The sets **U**, **V** and **W** denote the sets of variables whose value is in between the bounds, at their upper bound, and at their lower bound respectively.

The following lemma gives the representation for the feasibility cone $K(P - \bar{x})$.

Lemma 5.4 : *Let $\bar{x}$ be a solution to $\bar{P}$. Let {U, V, W} be a partition of variables as defined by (5.27). Then the feasibility cone $K(P - \bar{x})$ to $\bar{P}$ at $\bar{x}$ is given by the following system of linear constraints :*

$$\mathcal{A}y = 0, \tag{5.28}$$

$$y_j \geq 0 \quad \text{for all } j \in \mathbf{V}, \tag{5.29}$$

$$y_j \leq 0 \quad \text{for all } j \in \mathbf{W}. \tag{5.30}$$

Proof : By definition, y is a feasible direction to $\bar{P}$ at $\bar{x}$ if $\bar{x}+\theta y$, for sufficiently small $\theta > 0$, is a feasible solution to $\bar{P}$. Substituting $\bar{x}+\theta y$ for x in $\bar{P}$, we have $\mathcal{A}(\theta y) = 0$ and $l_j - \bar{x}_j \leq \theta y \leq u_j - \bar{x}_j$ for all $j = 1,2, \ldots, m$. $\mathcal{A}(\theta y) = 0$ if and only if $\mathcal{A}y = 0$. Since θ can be chosen arbitrarily small, the condition $l_j - \bar{x}_j \leq \theta y \leq u_j - \bar{x}_j$ for all $j = 1,2, \ldots, m$ is equivalent to the conditions (5.29) and (5.30). On the other hand, if y satisfies (5.28), (5.29) and (5.30), we have $\mathcal{A}(\theta y) = 0$ and $l_j - \bar{x}_j \leq \theta y \leq u_j - \bar{x}_j$ for all $j = 1,2, \ldots, m$ for sufficiently small $\theta > 0$. Hence y is a feasible direction of $\bar{P}$ at $\bar{x}$. ■

Lemma 5.5 : *The conormal cone $T(\bar{x}, \bar{P})$ is the set of all $t = (t_1, \ldots, t_m)$ satisfying the following linear constraints together with some π, an 1xn vector, and a set $\{\mu_j, j \in \mathbf{V} \cup \mathbf{W}\}$:*

$$t_j = \pi \mathcal{A}.j \qquad \text{for all } j \in \mathbf{U}, \tag{5.31}$$

$$t_j = \pi \mathcal{A}.j + \mu_j \qquad \text{for all } j \in \mathbf{V}, \tag{5.32}$$

$$t_j = \pi \mathcal{A}.j - \mu_j \qquad \text{for all } j \in \mathbf{W}, \tag{5.33}$$

$$\mu_j \geq 0 \qquad \text{for all } j \in \mathbf{V} \cup \mathbf{W} \tag{5.34}$$

where $\mathcal{A}.j$ denote the jth column of $\mathcal{A}$.

Proof : Recall that, by definition, a conormal t to $\bar{P}$ at $\bar{x}$ is a vector t such that $\bar{x}$ is an optimal solution to the following linear program :

$$\text{Minimize } t\,x \tag{5.35a}$$

$$\mathcal{A}x = b \tag{5.35b}$$

$$l_j \leq x_j \leq u_j, \quad j = 1,2, \ldots, m \tag{5.35c}$$

By the well-known complementary slackness conditions for bounded linear programming (see, for example, Murty [1976]), $\bar{x}$ is an optimal solution to the linear program (5.35) if and only if there exists a vector $\pi \in R^N$ satisfying the following conditions :

$$t_j = \pi \mathcal{A}.j \qquad \text{for all } j \in \mathbf{U} \tag{5.36}$$

$$t_j \geq \pi \mathcal{A}.j \qquad \text{for all } j \in \mathbf{V} \tag{5.37}$$

$$t_j \leq \pi \mathcal{A}.j \qquad \text{for all } j \in \mathbf{W} \tag{5.38}$$

These conditions are the same as (5.31)-(5.34), which proves the lemma. ■

When we use the representations of $T(\bar{x}, \bar{P})$ and $K(\bar{P} - \bar{x})$ developed in Lemma 5.4 and Lemma 5.5, the inverse linear programming problem (*ILP*) and its dual (*DILP*) become the following mathematical programs respectively.

Minimize $k(t-c)$ (5.39)

subject to

$$t_j = \pi \mathcal{A}.j \qquad \text{for all } j \in \mathbf{U}, \tag{5.40}$$

$$t_j \geq \pi \mathcal{A}.j + \mu_j \qquad \text{for all } j \in \mathbf{V}, \tag{5.41}$$

$$t_j = \pi \mathcal{A}.j - \mu_j \qquad \text{for all } j \in \mathbf{W}, \tag{5.42}$$

$$\mu_j \geq 0 \qquad \text{for all } j \in \mathbf{V} \cup \mathbf{W}. \tag{5.43}$$

Maximize $-cy$ (5.44)

subject to

$$\mathcal{A}y = 0, \tag{5.45}$$

$$y_j \geq 0 \qquad \text{for all } j \in \mathbf{V}, \tag{5.46}$$

$$y_j \leq 0 \qquad \text{for all } j \in \mathbf{W}, \tag{5.47}$$

$$y \in R^{\circ}. \tag{5.48}$$

Here k is an unorm and R° is the set generating its polar k°, that is, $R^{\circ} = \{y | k^{\circ}(y) \leq 1\}$.

Let $\bar{t} \in T(\bar{x}, \bar{P})$ and $\bar{y} \in K(\bar{P}-\bar{x})$. Then, for some $\bar{\pi}$ and $\{\mu : j \in \mathbf{V} \cup \mathbf{W}\}$, $\bar{t}$ satisfies (5.31)-(5.34). Therefore,

$$\bar{t}.\bar{y} = \pi \mathcal{A} \bar{y} + \sum_{j \in \mathbf{V}} \bar{\mu}_j \bar{y}_j - \sum_{j \in \mathbf{W}} \bar{\mu}_j \bar{\pi}_j$$

$$= \sum_{j \in \mathbf{V}} \bar{\mu}_j \bar{y}_j - \sum_{j \in \mathbf{W}} \bar{\mu}_j \bar{y}_j \quad (\text{since } \bar{\pi}\, \mathcal{A}\bar{y} = 0).$$

Since $\bar{\mu}_j \geq 0$ and $\bar{y}_j \geq 0$ for $j \in \mathbf{V}$ and $\bar{y}_j \leq 0$ for $j \in \mathbf{W}$, $\bar{t}.\bar{y} = \sum_{j \in \mathbf{V}} \bar{\mu}_j \bar{y}_j - \sum_{j \in \mathbf{W}} \bar{\mu}_j \bar{y}_j$ imply that $\bar{t}.\bar{y} = 0$ if and only if

$$\bar{\mu}_j \bar{y}_j = 0 \text{ for all } j \in \mathbf{V} \cup \mathbf{W}.$$

The following result is immediate from the above discussion and Theorem 5.8.

The inverse network flow problem (*IF*), given a unorm $k(x)$ on R^m, is to find a conormal $\bar{t}$ to F at $\bar{x}$ such that $k(\bar{t}-c) \leq k(t-c)$ for any conormal t (that is, with minimum k-variance from c). It is formulated as the following problem :

$$\text{Minimize } k(t-c) \tag{5.53}$$

subject to

$$t \in T(\bar{x}, F). \tag{5.54}$$

The optimal solution $\bar{t}$ to IF is called a minimal k-variance conormal to F at $\bar{x}$ from c.

Let k° be the polar of the unorm k. Let y be a feasible flow direction to F at $\bar{x}$, that is, $\bar{x} + \theta y$ is a feasible flow satisfying the flow conservation equations (5.51) and the flow constraints (5.52) for small values of θ. A feasible flow direction y is k°-normalized if $k^{\circ}(y) = 1$. The dual DIF of IF is defined as the problem of finding a maximal c-descent k°-normalized feasible direction to F at $\bar{x}$ or verifying there is no c-descent flow feasible direction to F at $\bar{x}$. The formulation of DIF is as follows :

$$\text{Maximize } -cy \tag{5.55}$$

subject to

$$y \in K(F - \bar{x}), \tag{5.56}$$

$$y \in R^{\circ}. \tag{5.57}$$

Here $K(F-\bar{x})$ is the feasibility cone to F at $\bar{x}$ and $R^{\circ}=\{y|k^{\circ}(y)\leq 1\}$.

The matrix representation of F is :

$$Nx = b, \tag{5.58}$$

$$0 \leq x \leq u. \tag{5.59}$$

Here b is the supply/demand vector and u is the capacity vector.

The matrix N is called the node-arc incidence matrix of the graph $G = (N, A)$. The columns of N are indexed by the arc set A and its rows by the node set N. The column corresponding to an arc (i, j) has 1 in row i and -1 in row j and zeros for rest of the entries. Let N_{ij} denote the column corresponding to the arc (i, j). Then,

$$\pi N_{ij} = \pi(i) - \pi(j) \tag{5.60}$$

Let $\bar{x}$ be a given flow and $\mathbf{U}$, $\mathbf{V}$, $\mathbf{W}$ denote the arc sets as defined below :

$$\mathbf{U} = \{(i, j) \in A \mid 0 < \bar{x}_{ij} < u_{ij}\}, \tag{5.61}$$

$$\mathbf{V} = \{(i, j) \in A \mid 0 = \bar{x}_{ij}\}, \tag{5.62}$$

$$\mathbf{W} = \{(i, j) \in A \mid \bar{x}_{ij} = u_{ij}\}. \tag{5.63}$$

The following polyhedral description of $T(\bar{x}, F)$ and $K(F - \bar{x})$ are immediate from Lemma 5.4 and Lemma 5.5.

The conormal cone $T(\bar{x}, F)$ by Lemma 5.5 is the set of all vectors $t = (t_{ij})$ satisfying the following constraints :

$$t_{ij} = \pi(i) - \pi(j) \qquad \text{for all } (i, j) \in \mathbf{U}, \tag{5.64}$$

$$t_{ij} = \pi(i) - \pi(j) + \mu_{ij} \qquad \text{for all } (i, j) \in \mathbf{V}, \tag{5.65}$$

$$t_{ij} = \pi(i) - \pi(j) - \mu_{ij} \qquad \text{for all } (i, j) \in \mathbf{W}, \tag{5.66}$$

$$\mu_{ij} \geq 0 \qquad \text{for all } (i, j) \in \mathbf{V} \cup \mathbf{W}. \tag{5.67}$$

Here $\pi(i)$ is a node potential associated with node i. Note that, by (5.60) the product of π with a column of N corresponding to (i, j) is $\pi(i) - \pi(j)$. Of course, the constraints (5.64)-(5.67) are the optimality conditions for flow $\bar{x}$ to be a minimum cost flow with respect to the cost t.

The feasibility cone $K(F - \bar{x})$, by Lemma 5.4, is the cone given below :

$$Ny = 0, \tag{5.68}$$

$$y_{ij} \geq 0 \qquad \text{for all } (i, j) \in \mathbf{V}, \tag{5.69}$$

$$y_{ij} \leq 0 \qquad \text{for all } (i, j) \in \mathbf{W}. \tag{5.70}$$

Thus a feasible direction to F at $\bar{x}$ is a circulation satisfying (5.68)-(5.70). We will call $K(F - \bar{x})$ also as the feasible circulation cone to F at $\bar{x}$.

Now we can restate the formulation of *IF* and *DIF* as follows. The formulation of *IF* is :

$$\text{Minimize } k(t-c) \tag{5.71a}$$

subject to

$$t_{ij} = \pi(i) - \pi(j) \qquad \text{for all } (i, j) \in \mathbf{U}, \tag{5.71b}$$

$$t_{ij} = \pi(i) - \pi(j) + \mu_{ij} \qquad \text{for all } (i, j) \in \mathbf{V}, \tag{5.71c}$$

$$t_{ij} = \pi(i) - \pi(j) - \mu_{ij} \qquad \text{for all } (i, j) \in \mathbf{W}, \tag{5.71d}$$

$$\mu_{ij} \geq 0 \qquad \text{for all } (i, j) \in \mathbf{A}. \tag{5.71e}$$

The formulation of *DIF* is :

$$\text{Maximize } -cy \tag{5.72a}$$

subject to

$$Ny = 0, \tag{5.72b}$$

$$y_{ij} \geq 0 \qquad \text{for all } (i, j) \in \mathbf{V}, \tag{5.72c}$$

$$y_{ij} \leq 0 \qquad \text{for all } (i, j) \in \mathbf{W}, \tag{5.72d}$$

$$y \in R^{\circ}, \tag{5.72e}$$

where $R^{\circ} = \{y \mid k^{\circ}(y) \leq 1\}$.

The inverse flow problem is a special case of the inverse linear programming problem. The results obtained in Theorem 5.2,

Theorem 5.4, Theorem 5.6 and Theorem 5.10 are summarized below for the inverse flow problem.

Theorem 5.11. *Let $c \in R^m$, and $\bar{x} \in F$. Let IF and DIF be the programs as defined by (5.71) and (5.72) respectively. Then,*

(a) Both IF and DIF have finite optimal solutions.

(b) The optimal values of IF and DIF are equal.

(c) Let $\bar{t}$ and $\bar{y}$ be the feasible solutions to IF and DIF respectively. Then they are optimal solutions to the respective problems if and only if they satisfy the following conditions :

(i) the linear function $(\bar{t}-c)y$ attains its maximum over R^o at $\bar{y}$, that is,

$$k(\bar{t}-c) = \delta^*(\bar{t}-c|R^o) = (\bar{t}-c)\bar{y}. \tag{5.73a}$$

(ii) $\mu_{ij}\, y_{ij} = 0$ *for all arcs* $(i, j) \in V \cup W$. (5.73b)

5.4.1 Euclidean Variance

An Euclidean inverse linear programing problem (EILP) finds a conormal t with minimum $||t-c||$, where

$$||x|| = \sqrt{\sum_{(i,j)\in A} x_{ij}^2}\ , \quad x \in R^m \tag{5.74}$$

The conormals we consider here are conormal to a polyhedron P at $\bar{x}$. Let

$$B = \{x \mid ||x|| \le 1\}. \tag{5.75}$$

The polar of $||\ ||$ is itself, and

$$||t|| = \gamma(t|B) = \delta^*(t|B) \tag{5.76}$$

Hence EILP is :

$$\text{Min } \{||t-c|| \ t \in T(\bar{x}, P)\} \tag{5.78}$$

Its dual DEILP is

$$\text{Max } \{-cy \mid y \in k(P-\bar{x}) \cap B\} \tag{5.79}$$

EILP has a unique optimum solution $\bar{t}$, since the strictly convex function $||t-c||$ has unique minimum over closed convex set (polyhedral cone) $T(\bar{x}, P)$. Let $\bar{y}$ be any optimal solution to the dual DEILP. Then, by Theorem 5.8, we have

(i) $||t-c|| = \delta^*(\bar{t}-c|B)$

$= \text{Max}\{(\bar{t}-c)y \mid y \in B\} = (\bar{t}-c)\bar{y}$, that is, $(\bar{t}-c)$ is normal to B at $\bar{y}$.

(ii) $\bar{t}.\bar{y} = 0$

By the duality in ILP (Theorem 5.6), we have

$$|| \bar{t}-c || = -c\bar{y} = \mu \text{ (say)}$$

When the problem is trivial, we have $\bar{t} = c$, $\bar{y} = 0$, when it is not trivial $\bar{y} \neq 0$ and $|| \bar{y} || = 1$ (see the proof of Theorem 5.3). In this case, any normal to B at $\bar{y}$ is a positive scalar multiple of $\bar{y}$, that is,

$$(\bar{t}-c) = \lambda \bar{y}, \qquad \lambda > 0.$$

Therefore, by (i), we have

$$|| t-c || = (\bar{t}-c)\bar{y} = \lambda \bar{y} \bar{y} = \lambda || \bar{y} ||^2 = \lambda.$$

Hence $\lambda = \mu$. Thus we have proved the following result.

Theorem 5.12 : *EILP and its dual have unique optimal solutions, say, $\bar{t}$ and $\bar{y}$ respectively. Then*

(a) $\bar{t} = c + \mu \bar{y}, \quad \mu = |\bar{t}-c| = -c\bar{y}$ (5.80a)

(b) $\bar{t}.\bar{y} = 0$ (5.80b)

Now consider the following minimization problems :

$$\text{Minimize } \{1/2 \, ||t-c||^2 : t \in T(\bar{x}, P)\} \tag{5.81}$$

$$\text{Minimize } \{1/2 \, ||y-c||^2 : y \in -K(P-\bar{x})\} \tag{5.82}$$

The problems (5.81) and EILP (5.78) are equivalent problems having the same unique optimal solution $\bar{t}$. The problem (5.82) also has an unique optimal solution. Let $\tilde{y}$ denote the optimal solution to (5.82). By Moreau decomposition theorem (see Theorem 31.5, Rockafellar [1970]), we have

$$c = \bar{t} + \tilde{y} \tag{5.83a}$$

and

$$\bar{t}.\tilde{y} = 0 \tag{5.83b}$$

Comparing (5.80) and (5.83),

$$\lambda \, \bar{y} = - \, \tilde{y} \tag{5.84}$$

where $\lambda = || \, \bar{t}-c \, || = -c.\bar{y}$.

Taking norm on both sides of (5.84), we have

$$\lambda = || \, \tilde{y} \, ||$$

Here

$$\bar{y} = - \frac{\tilde{y}}{|| \, \tilde{y} \, ||} , \tag{5.85}$$

and

$$|| \, \tilde{y} \, || = || \, \bar{t}-c \, || = -c\bar{y}. \tag{5.86}$$

Let $P = F$, where F is the system of flow constrains given by (5.51) and (5.52). Then by the representation of $K(F-\bar{x})$ given by (5.61)-(5.63), the problem

$$\text{Minimize } \{1/2 \, || \, y-c \, ||^2 : y \in -K(F-\bar{x}) \tag{5.87}$$

becomes the following quadratic cost flow problem (QCF)

$$\text{Minimize } 1/2 \sum_{(i,j)\in A} (y_{ij} - c_{ij})^2 \tag{5.88a}$$

$$Ny = 0, \tag{5.88b}$$

$$y_{ij} \le 0 \quad \text{for all } (i, j) \in V, \tag{5.88c}$$

$$y_{ij} \ge 0 \quad \text{for all } (i, j) \in W. \tag{5.88d}$$

We now conclude the following result from above discussion.

Theorem 5.13 : Let $\tilde{y}$ be the optimal solution to QCF defined by (5.88). Let $\bar{t}$ and $\bar{y}$ be the optimal solution to the Euclidean inverse flow problem (EIF) and its dual (DEIF). Then,

(a) $$|| \bar{t}-c || = - c\bar{y} = || \tilde{y} ||, \tag{5.89}$$

(b) $$\bar{y} = - \tilde{y}/|| \tilde{y} ||, \tag{5.90}$$

(c) $$\bar{t} = c - \tilde{y}. \tag{5.91}$$

5.4.2 Weighted Rectilinear Variance

In this section, we will take the objective function as weighted rectilinear variance in the inverse problem.

A weighted rectilinear variance of a vector t from c is

$$k_1(t-c) := \sum_{(i,j)\in A} \left(a_{ij}(t_{ij}-c_{ij})^+ + b_{ij}(t_{ij}-c_{ij})^- \right) \tag{5.92}$$

where a_{ij} and b_{ij} are positive real numbers ; for a real number t, $t^+ = \max\{0, t\}$ and $t^- = \max\{0, -t\}$.

The function

$$k_1(t) := \sum_{(i,j)\in A} (a_{ij}t_{ij}^+ + b_{ij}\, t_{ij}^-) \tag{5.93}$$

is a unorm and it is generated by the set

$$R_1 := \{ t \in R^m | \sum (a_{ij}t_{ij}^+ + b_{ij}\, t_{ij}^-) \le 1 \} \tag{5.94}$$

That is,

$$k_1(t) = \gamma(t|R_1) \tag{5.95}$$

The polar of R_1 is the set

$$R_\infty = \{y \in R^m | -b_{ij} \le y_{ij} \le a_{ij}\} \tag{5.96}$$

and R_∞ generates the following unorm :

$$k_\infty(t) := \max \left\{ \frac{y_{ij}^+}{a_{ij}} , \frac{y_{ij}^-}{b_{ij}} \mid (i, j) \in A \right\} \tag{5.97}$$

Thus,

$$k_\infty(t) = \gamma(t|R_\infty) \tag{5.98}$$

The unorms k and k_∞ are polar to each other.

The following lemma is a direct application of the Proposition 5.1.

Lemma 5.6 : (a) $k_1(t) = \delta^*(t|R_\infty)$ and its conjugate is $\delta(t|R_\infty)$.

(b) $k_\infty(t) = \delta^*(t|R_1)$ and its conjugate is $\delta(t|R_1)$.

Now we state the weighted rectilinear inverse flow problem, (WRIF):

$$\text{Minimize } k_1(t-c) \tag{5.99}$$

subject to

$$\pi(i) - \pi(j) = t_{ij} \qquad \text{for } (i, j) \in U, \tag{5.100a}$$

$$\pi(i) - \pi(j) + \mu_{ij} = t_{ij} \qquad \text{for } (i, j) \in V, \tag{5.100b}$$

$$\pi(i) - \pi(j) - \mu_{ij} = t_{ij} \qquad \text{for } (i, j) \in W. \tag{5.100c}$$

$$\mu_{ij} \ge 0 \tag{5.100d}$$

An optimal solution $\bar{t}$ to WRIF will be called a minimal weighted rectilinear conormal variance to F a $\bar{x}$ from c.

The dual DWRIF of WRIF is to find a maximal c-descent normalized feasible direction to F at $\bar{x}$ with respect to unorm k_∞. Thus DWRIF is the following problem :

$$\text{Maximize } -cy \tag{5.101}$$

subject to

$$Ny = 0 \tag{5.102a}$$

$$y_{ij} \geq 0 \quad \text{for } (i, j) \in V, \tag{5.102b}$$

$$y_{ij} \leq 0 \quad \text{for } (i, j) \in W. \tag{5.102c}$$

$$y \in R_\infty. \tag{5.102d}$$

The description of R_∞ given in (5.96) imply that the constraints (5.100b), (5.100c) and (5.100d) in DWRIF can be replaced by (5.102c), (5.102d) and (5.102e) in the following formulation of DWRIF.

$$\text{Maximize } -cy \tag{5.103}$$

subject to

$$Ny = 0 \tag{5.104a}$$

$$b_{ij} \leq y_{ij} \leq a_{ij} \quad \text{for } (i, j) \in U, \tag{5.104b}$$

$$0 \leq y_{ij} \leq a_{ij} \quad \text{for } (i, j) \in V, \tag{5.104c}$$

$$-b_{ij} \leq y_{ij} \leq 0 \quad \text{for } (i, j) \in W. \tag{5.104d}$$

We can interpret $-c_{ij}$ as the profit. Note that maximizing the profit $-cy$ is equivalent to minimizing the cost cy. Thus, the representation of WRIF given by (5.103)-(5.104) is a minimum cost (maximum profit) circulation on the given network $G = (N, A)$ with (finite) bounds given by (5.104b)-(5.104c).

Let $\bar{y}$ be an optimal solution to WRIF (5.103)-(5.104), that is, a maximal c-descent feasible direction to F at $\bar{x}$. Then by the complementary slackness for linear programming, there exists an

optimal node potential π satisfying the following conditions:

$$\pi(i) - \pi(j) = c_{ij} \text{ if } \bar{\ell}_{ij} < \bar{y}_{ij} < \bar{u}_{ij}\ , \tag{5.105a}$$

$$\text{if} \quad \pi(i) - \pi(j) < c_{ij} \text{ then } \bar{\ell}_{ij} = \bar{y}_{ij}\ , \tag{5.105b}$$

$$\text{if} \quad \pi(i) - \pi(j) > c_{ij} \text{ then } \bar{y}_{ij} = \bar{u}_{ij}\ , \tag{5.105c}$$

where $\bar{\ell}_{ij}$ is $-b_{ij}$ if $(i, j) \in U \cup W$, 0 otherwise, and $\bar{u}_{ij}$ is a_{ij} if $(i, j) \in U \cup V$, 0 otherwise.

Let π be such a potential, and let

$$\bar{t}_{ij} = \pi(i) - \pi(j) \qquad \text{for } (i, j) \in U \tag{5.106a}$$

$$\bar{t}_{ij} = \max \{\pi(i) - \pi(j), c_{ij}\} \quad \text{for } (i, j) \in V \tag{5.106b}$$

$$\bar{t}_{ij} = \min \{\pi(i) - \pi(j), c_{ij}\} \quad \text{for } (i, j) \in W \tag{5.106c}$$

Define

$$\mu_{ij} = |t_{ij} - c_{ij}| \quad \text{for } (i, j) \in V \cup W \tag{5.107}$$

Then $\bar{t}$, together with π and μ_{ij}'s defined by (5.107) satisfy (5.100). Hence $\bar{t}$ is a conormal to F at $\bar{x}$.

Suppose $\mu_{ij} > 0$. If $(i, j) \in V$, then $\pi(i) - \pi(j) < c_{ij}$ by (5.106) and hence by (5.105) $\bar{y}_{ij} = \bar{\ell}_{ij} = 0$. If $(i, j) \in W$, then $\pi(i) - \pi(j) > c_{ij}$ by (5.106), and hence by (5.106) $\bar{y}_{ij} = \bar{u}_{ij} = 0$. Therefore, we have

$$\mu_{ij}\, \bar{y}_{ij} = 0 \text{ for all arcs } (i, j) \in V \cup W \tag{5.108}$$

Now consider an arc (i, j) and let $\bar{t}_{ij} > c_{ij}$. Then arc (i, j) is in either U or V; in both cases, $\bar{u}_{ij} = a_{ij}$. By definition of $\bar{t}_{ij}$, $\bar{t}_{ij} = \pi(i) - \pi(j)$. Therefore $\pi(i) - \pi(j) = \bar{t}_{ij} > c_{ij}$ implies $\bar{y}_{ij} = \bar{u}_{ij}$ (by (5.106c)). Hence $\bar{y}_{ij} = a_{ij}$. Therefore, we have, if $t_{ij} > c_{ij}$,

$$a_{ij}\,(\bar{t}_{ij} - c_{ij})^{+} + b_{ij}\,(\bar{t}_{ij} - c_{ij})^{-} = a_{ij}\,(\bar{t}_{ij} - c_{ij}) = \bar{y}_{ij}\,(\bar{t}_{ij} - c_{ij})$$

By similar arguments, if $\bar{t}_{ij} < c_{ij}$, then arc (i, j) is either in V or W, and $\bar{y}_{ij} = -b_{ij}$. Therefore, if $\bar{t}_{ij} > c_{ij}$

$$a_{ij}\,(\bar{t}_{ij} - c_{ij})^{+} + b_{ij}\,(\bar{t}_{ij} - c_{ij})^{-} = b_{ij}\,(\bar{t}_{ij} - c_{ij}) = -b_{ij}\,(\bar{t}_{ij} - c_{ij}) = \bar{y}_{ij}\,(\bar{t}_{ij} - c_{ij})$$

Therefore, we have

$$a_{ij}\,(\bar{t}_{ij} - c_{ij})^{+} + b_{ij}\,(\bar{t}_{ij} - c_{ij})^{-} = \bar{y}_{ij}\,(\bar{t}_{ij} - c_{ij}) \quad \text{for all } (i, j) \in A, \qquad (5.109)$$

since (5.109) holds obviously for the case $\bar{t}_{ij} = c_{ij}$. Summing right and left hand sides of (5.109) over all arcs, we have

$$k_1\,(\bar{t} - c) = \sum_{(i,j)\in A} \bar{y}_{ij}\,(\bar{t}_{ij} - c_{ij}) = (\bar{t} - c)\bar{y} \qquad (5.110)$$

Since $\bar{t}_{ij}$'s satisfy (5.100),

$$\bar{t}.\bar{y} = \Sigma\, \bar{t}_{ij}\, \bar{y}_{ij} = \pi\, N\, \bar{y} + \sum_{(i,j)\in V} \mu_{ij}\, \bar{y}_{ij} - \sum_{(i,j)\in W} \mu_{ij}\, \bar{y}_{ij}$$

Since $N.\bar{y} = 0$, and $\mu_{ij}\, \bar{y}_{ij} = 0$ for $(i, j) \in V \cup W$ by (5.106), we have $\bar{t}.\bar{y} = 0$. therefore, by (5.110) $k_1(\bar{t}-c) = -c\bar{y}$. Hence, by Theorem 5.7, $\bar{t}$ is an optimal solution to WRIF, that is, a minimal k_1-variance conormal to F at $\bar{x}$ from c. From the above discussion, we conclude the following theorem.

Theorem 5.14 : *The dual of WRIF (DWRIF) can be formulated as the minimum cost (maximum profit) circulation problem given by (5.103)-(5.104). Let $\bar{\pi}$ be an optimum node potential corresponding to a minimum cost circulation $\bar{y}$ to (5.103)-(5.104). Then $\bar{t}$ given by (5.106) is minimal k_1-variance to F at $\bar{x}$ from c. Further, $k_1(\bar{t}-c) = -c\bar{y}$.*

Recall that WRIF is trivial when $\bar{t} = c$; this is the case if and only if $\bar{x}$ is a minimal cost flow to F (defined by (5.51)-(5.52)). In this case, $\bar{y} = 0$ is an optimal solution to DWRIF.

When $\bar{x}$ is not a minimum cost flow to (5.51)-(5.52), then $k_\infty(\bar{y}) = 1$ by Theorem 5.3. That is, $\bar{y}$ is a circulation on $G = (N, A)$ with $\bar{y}_{ij} = a_{ij}$ or $-b_{ij}$ for at least one arc. Further, when a_{ij}'s and b_{ij}'s are integers, the minimum cost circulation problem defined by (5.103)-(5.104) always has integral basic optimum solution (see Property 2.1); when c_{ij}'s are integers, the minimum cost circulation problem has an integral optimal node potentials corresponding to any basic optimum solution (see, Property 2.2). Theorem 5.14 together with (5.106) imply that WRIF defined by (5.99)-(5.100) has integral optimum solutions. We summarize the above discussion in the following theorem.

Theorem 5.15 : *Suppose $\bar{x}$ is not a minimum cost flow to (5.51)-(5.52). Then :*
(a) any maximal c-descent feasible direction normalized by $k_\infty(t-c)$ has the property that for at least one arc (i, j), $\bar{y}_{ij} = a_{ij}$ or $-b_{ij}$. This is a minimum cost circulation to (5.103)-(5.104).

(b) When a_{ij}'s and b_{ij}'s in $k_1(t-c)$ are integers, we have at least one integral maximal c-descent normalized feasible direction with respect to $k_\infty(t-c)$ which is a basic integral minimum cost circulation to (5.103)-(5.104).

(c) When c_{ij}'s are integers, we have at least one integral minimum k_1-variance conormal from c.

A Special Case : Rectilinear Variance

The rectilinear variance of a vector t from c is

$$||t-c||_1 = \sum_{(i,j)\in A} |t_{ij}-c_{ij}| \tag{5.111}$$

Note that $||t-c||_1$ is a special case of $k_1(t-c)$ with $a_{ij} = b_{ij} = 1$.

Suppose the given $\bar{x}$ is not a minimum cost flow. Then, by Theorem 5.15, we have maximal c-descent normalized feasible direction with respect to $||\ ||_\infty$ which is a basic integral minimum cost circulation to the following problem :

Maximize $-cy$ (Minimize cy)

subject to

$$\begin{aligned} &Ny = 0, && \\ &-1 \le y_{ij} \le 1 && \text{for all } (i,\ j) \in U, \\ &0 \le y_{ij} \le 1 && \text{for all } (i,\ j) \in V, \\ &-1 \le y_{ij} \le 0 && \text{for all } (i,\ j) \in W. \end{aligned} \tag{5.112}$$

Any integral circulation to (5.112) can be decomposed to at most m integral flows on at most m cycles in G by the flow decomposition theorem (see, the discussion in Section 4.2). Since the flow bounds in (5.112) are only between -1 and 1, any integral

circulation has y_{ij} = -1, 0, or 1 for each arc $(i, j) \in A$. Consequently, integral cyclic flow in the above decomposition are unit flows and the corresponding cycles are disjoint. These are augmenting cycles with respect to the flow $\bar{x}$ and the cost of any integral circulation to (5.112) is the sum of costs of these disjoint cycles. Thus an optimum integral circulation to (5.112) is a minimum cost disjoint augmenting cycles (that is, a unit circulation on these cycles) with respect to the flow. Thus we have proved the following theorem.

Theorem 5.16 : *(a) A minimum cost disjoint augmenting cycles with respect to the flow $\bar{x}$ is a maximal c-descent normalized feasible direction with respect to $||\ ||_\infty$.*
(b) The minimum k-variance from c over all conormals at $\bar{x}$ with respect to $||\ ||_1$ is negative of the cost of a minimum cost disjoint augmenting cycles with respect to $\bar{x}$.

5.4.3 Weighted Maximum Variance

A weighted maximum variance of a vector t from c is defined by

$$k_\infty(t-c) = \max_{(i,j)\in A} \left\{ \frac{(t_{ij}-c_{ij})^+}{a_{ij}} , \frac{(t_{ij}-c_{ij})^-}{b_{ij}} \right\} \qquad (5.113)$$

By Lemma 5.6,

$$k_\infty(t) = \delta^*(t|R_1) \qquad (5.114)$$

where R_1 is given by

$$R_1 = \left\{ y \Big| \sum_{(i,j)\in A} (a_{ij}\, y_{ij}^{+} + b_{ij}\, y_{ij}^{-}) \le 1 \right\}. \qquad (5.117)$$

The set R_1 is the polar of R_∞ generating unorm $k_\infty(t)$, and generates unorm $k_1(t)$.

The weighted maximum inverse flow problem (WMIF) is to find a minimal weighted maximum variance conormal to F at $\bar{x}$ from c. Its dual (DWMIF) is to find a maximal c-descent normalized feasible direction to F at $\bar{x}$ with respect to unorm $k_1(t)$. Thus the following formulations of WMIF and DWMIF follows directly from (5.71) and (5.72).

The weighted maximum inverse network flow problem WMIF is :

$$\text{Minimize} \quad \max_{(i,j)\in A} \left\{ \frac{(t_{ij}-c_{ij})^+}{a_{ij}} , \frac{(t_{ij}-c_{ij})^-}{b_{ij}} \right\} \tag{5.116a}$$

subject to

$$\pi(i) - \pi(j) = t_{ij} \qquad \text{for all } (i, j) \in U, \tag{5.116b}$$

$$\pi(i) - \pi(j) \le t_{ij} \qquad \text{for all } (i, j) \in V, \tag{5.166c}$$

$$\pi(i) - \pi(j) \ge t_{ij} \qquad \text{for all } (i, j) \in W. \tag{5.116d}$$

The dual DWMIF of the weighted maximum inverse network flow problem is :

$$\text{Maximize} \quad -\sum_{(i,j)\in A} c_{ij}\, y_{ij} \tag{5.117a}$$

subject to

$$Ny = 0, \tag{5.117b}$$

$$y_{ij} \ge 0 \qquad \text{for } (i, j) \in V, \tag{5.117c}$$

$$y_{ij} \le 0 \qquad \text{for } (i, j) \in W, \tag{5.117d}$$

$$\sum_{(i,j)\in A} (a_{ij}y_{ij}^{+} + b_{ij}y_{ij}^{-}) \le 1. \tag{5.117e}$$

The dual can be transformed to a linear programming problem with nonnegative variables on a modified network $\hat{G} = (N, \hat{A})$. For each arc (i, j) in $U \cup V$ (the set of arcs on which flow can be

increased), introduce an arc (i, j) in $\hat{A}$ with cost c_{ij} and weight a_{ij}; for each arc (i, j) in $U \cup W$ (the set of arcs on which flow can be decreased), introduce an arc (j, i) with cost $-c_{ij}$ and weight b_{ij}. Let us denote the cost and the weights, as defined above, of arcs in $\hat{A}$ by $\hat{c}_{ij}$ and $\hat{w}_{ij}$ respectively. The dual problem can be rewritten equivalently with respect to the network $\hat{G} = (N, \hat{A})$ as follows :

$$\text{- Minimize } \sum_{(i,j)\in\hat{A}} \hat{c}_{ij} z_{ij} \quad (\text{Maximize } - \sum_{(i,j)\in\hat{A}} \hat{c}_{ij} z_{ij}) \tag{5.118a}$$

subject to

$$\hat{N}z = 0, \tag{5.118b}$$

$$z_{ij} \geq 0 \qquad \text{for all } (i, j) \in \hat{A}, \tag{5.118c}$$

$$\sum_{(i,j)\in\hat{A}} \hat{w}_{ij} z_{ij} \leq 1. \tag{5.118d}$$

Here $\hat{N}$ is the node arc incidence matrix of the graph $\hat{G}$.

The dual DWMIF is equivalent to solving the minimum cost-to-weight ratio cycle $\bar{\Gamma}$ in $\hat{G}$, where cost-to-weight ratio of a directed cycle Γ in $\bar{G}$ is defined by

$$\mu(\Gamma) = \frac{\sum_{(i,j)\in\Gamma} \hat{c}_{ij}}{\sum_{(i,j)\in\Gamma} \hat{w}_{ij}} . \tag{5.119}$$

Note that $a_{ij} > 0$ and $b_{ij} > 0$ for each arc $(i, j) \in A$.

By definition of $\hat{w}_{ij}$, the weight $\sum_{(i,j)\in\Gamma} \hat{w}_{ij} > 0$ for every directed cycle Γ in $\bar{G}$.

Let Γ^* denote any minimum cost-to-weight ratio cycle in $\hat{G}$. Then we have

$$\mu(\Gamma^*) = \frac{C(\Gamma^*)}{W(\Gamma^*)}, \tag{5.120}$$

and $$\mu(\Gamma^*) \leq \frac{C(\Gamma)}{W(\Gamma)} \text{ for any directed cycle } \Gamma \text{ in } \hat{G}. \tag{5.121}$$

Let

$$\bar{\mu} = -\mu(\Gamma^*). \tag{5.122}$$

Then, for any directed cycle Γ in $\hat{G}$, by (5.121), we have $C(\Gamma) + \bar{\mu}\, W(\Gamma) \geq 0$, that is, $\sum_{(i,j)\in\Gamma} (\hat{c}_{ij} + \bar{\mu}\, \hat{w}_{ij}) \geq 0$. Since there is no negative directed cycled in $\hat{G}$ with respect to costs $\hat{c}_{ij} + \bar{\mu}\, \hat{w}_{ij}$, there exist node potentials π satisfying the following conditions:

$$\pi(i) - \pi(j) \leq \hat{c}_{ij} + \bar{\mu}\, \hat{w}_{ij} \text{ for all } (i, j) \in \hat{G} \tag{5.123}$$

Suppose $\pi(i) - \pi(j) < \hat{c}_{ij} + \bar{\mu}\, \hat{w}_{ij}$ for at least one arc (i, j) in Γ^*. Then, summing the right hand sides and the left hand sides of (5.123) over all arcs in Γ^*, we have $0 < C(\Gamma^*) + \bar{\mu}\, w\ (\Gamma^*)$ which contradicts (5.120) together with (5.122). Hence

$$\pi(i) - \pi(j) = \hat{c}_{ij} + \bar{\mu}\, \hat{w}_{ij} \text{ for all } (i, j) \in \Gamma^*. \tag{5.124}$$

Let $\bar{t}_{ij} = \pi(i) - \pi(j)$ for each arc $(i, j) \in A$. Since $\hat{c}_{ij} = c_{ij}$ and $\hat{w}_{ij} = a_{ij}$ if $(i, j) \in A$. and $\hat{c}_{ij} = -c_{ij}$ and $\hat{w}_{ij} = b_{ij}$ if $(j, i) \in A$, (5.123) implies that

$$-\bar{\mu}\, b_{ij} \leq t_{ij} - c_{ij} \leq \bar{\mu}.a_{ij}$$

for each arc $(i, j) \in A$. Since $a_{ij} > 0$ and $b_{ij} > 0$, this implies

$$\max_{(i,j)\in A} \left\{ \frac{(\bar{t}_{ij}-c_{ij})^+}{a_{ij}}, \frac{(\bar{t}_{ij}-c_{ij})^-}{b_{ij}} \right\} \leq \bar{\mu}$$

Further, for any arc $(i, j) \in \Gamma^*$, if $(i, j) \in A$, by (5.124), then

$\frac{(\bar{t}_{ij}-c_{ij})^+}{a_{ij}} = \bar{\mu}$; else we have $(j, i) \in A$ and $\frac{(\bar{t}_{ij}-c_{ij})^-}{b_{ij}} = \bar{\mu}$.

Hence,

$$(\bar{t}-c) = \max_{(i,j)\in A}\left\{ \frac{(\bar{t}_{ij}-c_{ij})^+}{a_{ij}} , \frac{(\bar{t}_{ij}-c_{ij})^-}{b_{ij}} \right\} = \bar{\mu} .$$

Further, $\bar{t}$ together with π obviously satisfies (5.116).

Let $\bar{z}$ be the circulation on $\hat{G}$ defined by

$$\bar{z}_{ij} = \begin{cases} 1/W(\Gamma^*) & \text{if } (i, j) \in \Gamma^* \\ 0 & \text{otherwise} \end{cases}$$

then $\bar{z}$ is feasible circulation to (5.118), since

$$\sum_{(i,j)\in\hat{A}} \hat{w}_{ij}\, \bar{z}_{ij} = \sum_{(i,j)\in\Gamma^*} \hat{w}_{ij}\, \bar{z}_{ij} = \frac{W(\Gamma^*)}{W(\Gamma^*)} = 1.$$

Further,

$$-\hat{c}.\bar{z} = \sum_{(i,j)\in\Gamma^*} \hat{c}_{ij}.\bar{z}_{ij} = -\frac{C(\Gamma^*)}{W(\Gamma^*)} = \bar{\mu}$$

since $\bar{t}$ is a feasible solution to WMIF and $\bar{z}$ is a feasible solution to (5.118), which is equivalent to DWMIF, and

$$k(\bar{t}-c) = -\hat{c}.\bar{z} = \bar{\mu},$$

by Theorem 5.7, $\bar{t}$ is an optimal solution to WMIF and $\bar{\mu}$ is the optimal value of WMIF (and DWMIF).

When $\bar{x}$ is a minimum cost flow, $\bar{t} = c$ and $\bar{z} = 0$ are optimal solutions to WMIF and (5.118) respectively. Thus, in this case, zero is the optimal value for both WMIF and its dual.

When $\bar{x}$ is not an optimal flow, $\mu(\Gamma^*) < 0$; $\bar{\mu} = -\mu(\Gamma^*) > 0$. Hence the optimal value of WMIF and its dual DWMIF is $\max\{0, -\mu(\Gamma^*)\}$.

Let Γ denote a augmenting cycle with respect to the flow $\bar{x}$ in $\bar{G}$, and let $\overline{\Gamma}$ and $\underline{\Gamma}$ denote the sets of forward and backward arcs in Γ respectively. The cost $C(\Gamma)$ and weight $W(\Gamma)$ of the cycle Γ are defined by

$$C(\Gamma) = \sum_{(i,j)\in\overline{\Gamma}} c_{ij} - \sum_{(i,j)\in\underline{\Gamma}} c_{ij} \tag{5.125}$$

$$W(\Gamma) = \sum_{(i,j)\in\overline{\Gamma}} a_{ij} - \sum_{(i,j)\in\underline{\Gamma}} b_{ij} \tag{5.126}$$

$$\mu(\Gamma) = C(\Gamma)/W(\Gamma) \tag{5.127}$$

Note that Γ corresponds to the directed cycle Γ^1 in $\hat{G}$ which consists of arcs in $\overline{\Gamma}$ and the reversal of the arcs in $\underline{\Gamma}$ and the cost-to- weight ratio of the cycle Γ, $\mu(\Gamma) = \frac{C(\Gamma)}{W(\Gamma)}$ is same as the cycle $\mu(\Gamma^1)$ as defined by (5.119). (Note that the construction of $\hat{G}$ is same as $G(\bar{x})$ except that arcs in $\hat{G}$ have weights and no residual capacities). A cyclic flow on Γ is a cyclic flow Γ^1. This observation with our discussion implies the following theorem.

Theorem 5.17 : *(a) The optimal value of WMIF and its dual DWMIF is* $\max\{0, -\mu(\Gamma^*)\}$, *where* Γ^* *is a minimum cost-to-weight ratio augmenting cycle with respect to the flow* $\bar{x}$ *on G, and the cost-to-weight ratio* $\mu(\Gamma)$ *of any augmenting cycle* Γ *is defined by (5.127).*

(b) Let π *be any node potential satisfying (5.123) and (5.122). Then* $(\bar{t}_{ij} = \pi(i) - \pi(j) : (i, j) \in A)$ *is a minimum weighted maximum variance conormal to F at* $\bar{x}$ *from c.*

(c) The cyclic flow of value $1/W(\Gamma^*)$, *where* $W(\Gamma^*)$ *is a maximal c-descent feasible direction to F at* $\bar{x}$ *with respect to the unorm* k_1 *(the polar of* k_∞*).*

Special Case : Maximum Variance

The maximum variance of t from c is

$$||t-c||_\infty = \max_{(i,j)\in A} |t_{ij} - c_{ij}|\}.$$

Thus $||t-c||_\infty$ is a special case of $k_\infty(t-c)$ with $a_{ij} = b_{ij} = 1$.

In this case, the weight $W(\Gamma)$ of an augmenting cycle Γ becomes the number of arcs in Γ denoted by $|\Gamma|$. The cost-to-weight ratio $\mu(\Gamma)$ of Γ becomes the mean cost of cycle Γ defined by

$$\mu_o(\Gamma) = \frac{\sum_{(i,j)\in\overline{\Gamma}} c_{ij} - \sum_{(i,j)\in\underline{\Gamma}} c_{ij}}{|\Gamma|} \qquad (5.128)$$

The following result is the direct corollary of Theorem 5.17.

Theorem 5.18 : The optimal value of MIF and its dual DMIF is $\max\{0, -\mu_o(\Gamma^*)\}$, where Γ^* is the minimum mean cost augmenting cycle with respect to the flow $\overline{x}$.

5.5 Inverse Spanning Tree Problem

Consider a undirected graph $G = (N, A)$. Let T^o be a spanning tree in G. Let c be a given cost vector, and k be an unorm. The inverse spanning tree problem (IST) is to find a cost vector t such that $k(t-c)$ is minimum and T^o is a minimum spanning tree with respect to cost t.

The co-normal cone for T^o

The co-normal cone of T^o is the set of all vectors t such that T^o is a minimum spanning tree with respect to the cost t. Thus the co-normal cone of T^o is the feasibility region of the inverse spanning tree problem.

We derive the representation for the co-normal cone of T^O from the following optimality conditions (For example, see Ahuja et al. (1993), Theorem 13.1.).

Cut Optimality Conditions : Let $[S_k, S_l]$ denote the cut obtained by removing a tree arc $(k, l) \in T^O$. The spanning tree T^O is a minimum spanning tree with respect to cost vector t if and only if it satisfies the following optimality conditions :

$$t_{ij} \geq t_{kl} \quad \text{for every } (i, j) \in [S_k, S_l] \text{ and for every } (k, l) \in T^O \tag{5.129}$$

We now present the optimality conditions in (5.129) as network dual constraints. Let us index the non-tree arcs by $q_1 = (i_1, j_1)$, $q_2 = (i_2, j_2)$, ..., $q_{m-n+1} = (i_{m-n+1}, j_{m-n+1})$ and tree arcs by $\gamma_1 = (i_{m-n+2}, j_{m-n+2})$, $\gamma_2 = (i_{m-n+3}, j_{m-n+3})$, ..., $\gamma_{n-1} = (i_m, j_m)$. We define the cut-graph $G(T^O) = (N(T^O), A(T^O))$ as follows. We define the node-set $N(T^O) = N_1(T^O) \cup N_2(T^O)$, where $N_1(T^O) = \{q_1, \ldots, q_{m-n+1}\}$ represents the non-tree arcs, and $N_2(T^O) = \{\gamma_1, \ldots, \gamma_{n-1}\}$ represents the tree-arcs. We introduce arcs in $G(T^O)$ as follows. Let $(k, l) \in T^O$ and $(k, l) = r_j$, we denote the cut $[S_k, S_l]$ by $[S_{r_j}, \bar{S}_{r_j}]$. Now, for each arc, $r_j = (k, l) \in T^O$ and for each tree arc $q_i = (p, q) \in [S_{r_j}, \bar{S}_{r_j}]$, we introduce an arc (q_i, γ_j) in $G(T^O)$. The graph $G(T^O)$ constructed as above is a bipartite graph.

To simplify the notations, we let $N_1(T^O) = \{q_1, q_2, \ldots, q_{m-n+1}\} = \{1, 2, \ldots, m-n+1\}$, and $N_2(T^O) = \{r_1, r_2, \ldots, r_{n-1}\} = \{1, 2, \ldots, n-1\}$. We denote the arc (q_i, r_j) in $G(T^O)$ by (i, j).

From the construction of $G(T^O)$, we have the following result, which is an alternate statement of cut-optimality conditions.

Theorem 5.19 : *The spanning tree T^O is a minimum spanning tree with respect to a cost vector t if and only if it satisfies the following optimality conditions :*

$$u_i - v_j \geq 0 \quad \text{for every arc } (i, j) \in G(T^O). \tag{5.130}$$

Here $t = (t_{i_1,j_1}, \ldots, t_{i_m,j_m}) = (u_1, \ldots, u_{m-n+1}, v_1, \ldots, v_{n-1})$.

Proof : By construction, each arc $(k, l) \in T$, has a node in $N_2(T^O)$ and each arc $(i, j) \in [S_k, S_l]$ has a node i in $N_1(T^O)$. Conversely, each arc $(i, j) \in G(T^O)$ represents a relation $(i, j) \in [S_k, S_l]$ for some arc $(k, l) \in T^O$. Thus, the condition $t_{ij} \geq t_{kl}$ for $(i, j) \in (S_k, S_l)$ is equivalent to $u_i - v_j \geq 0$ for $(i, j) \in G(T^O)$.

Corollary 5.18.1 : *The co-normal cone $T(T^O)$ of T^O is the set,*

$$T(T^O) = \{t: u_i - v_j \geq 0 \text{ or exactly } (i, j) \in G(T^O)\},$$

where $t = (u_1, u_2, \ldots, u_{m-n+1}, v_1, \ldots, v_{n-1})$.

In matrix form,

$$T(T^O) = \{t: N_{T^O}^T \, t \geq 0\}.$$

where N_{T^O} is the node-arc incidence matrix of $G(T^O)$.

Thus the inverse spanning tree problem (IST) is :

$$\text{Minimize } k(t-c) \tag{5.131a}$$

subject to

$$t \in T(T^O), \tag{5.131b}$$

where $k(t)$ is a unorm.

The dual DIST of IST is:

$$\text{Maximize } -cy \tag{5.132a}$$

subject to

$$y \in [T(T^O)]^*, \tag{5.132b}$$

$$y \in R^O, \tag{5.132c}$$

where R^O is the set generating the polar k^* of k and $[T(T^O)]^*$ is the copolar of $T(T^O)$.

Theorem 5.19 : *The copolar $[T(T^O)]^*$ is the set*

$$[T(T^O)]^* = \{ y | y = N_{T^O} x,\ x \geq 0 \}.$$

Proof : $T(T^O) = \{t: a_j\ t \geq 0\}$, where a_j are column vectors of N_{T^O}, and j is the index set of column of N_{T^O}. By Lemma 5.2, its copolar is the cone $\{a_j | j \in J\} = \{y |\ y = N_{T^O} x,\ x \geq 0\}$. ■

Now we use the representation of $T(T^O)$ and its polar $[T(T^O)]^*$, in the formulation of the inverse spanning tree problem and its dual.

The inverse spanning tree problem (IST) has the following formulation:

$$\text{Minimize } k(t-c) \tag{5.133a}$$

subject to

$$N^T_{T^O}\ t \geq 0. \tag{5.133b}$$

The dual DIST of IST is:

$$\text{Maximize } -cy \tag{5.134a}$$

subject to

$$N_{T^O}.x = y, \tag{5.134b}$$

$$x \geq 0, \tag{5.134c}$$

$$y \in R^O. \tag{5.134d}$$

The following duality results follow directly from Theorem 5.6 and Theorem 5.7.

Theorem 5.20 : (Duality results in Inverse Spanning Tree Problem).

Let T^o be a spanning tree and c be a given cost vector. The optimal values of (IST) and (DIST) are equal, that is,

$$\min\{\, k(t-c):\ N^T_{T^o}\, t \geq 0 \,\}$$

$$= \max\{\, -cy:\ N_{T^o}\, x = y,\ x \geq 0,\ y \in R^o \,\} \qquad (5.135)$$

Let $\bar{t}$ and $\bar{y}$ be the feasible solutions to IST and DIST respectively. Then they are optimal to the respective problems if and only if

$$k(\bar{t}-c) = -c\bar{y}. \qquad (5.136)$$

5.5.1 Weighted Rectilinear Variance

Here we take the weighted rectilinear variance defined by

$$k_1(t-c) := \sum_{(i,j)\in A} \left(a_{ij}(t_{ij}-c_{ij})^+ + b_{ij}(t_{ij}-c_{ij})^- \right) \qquad (5.137)$$

as the objective function in (IST). We call the corresponding IST as the weighted rectilinear inverse spanning tree problem (WRIST).

We have,

$$k_1(t) = \sum_{(i,j)\in A} (a_{ij}t_{ij}^+ + b_{ij}t_{ij}^-) \qquad (5.138)$$

and its polar,

$$k_\infty(y) = \max_{(i,j)\in A} \left\{ \frac{y_{ij}^+}{a_{ij}} , \frac{y_{ij}^-}{b_{ij}} \right\} \qquad (5.139)$$

is generated by the set

$$R_o = R_\infty = \{-b_{ij} \leq y_{ij} \leq a_{ij} : (i, j) \in A\} \qquad (5.140)$$

Consequently, the dual DWRIST of WRIST is

Maximize -cy (- Minimize cy) (5.141a)

subject to

$$y = N_{T^0}\, x, \tag{5.141b}$$

$$-b_{ij} \le y_{ij} \le a_{ij} \quad \text{for every } (i, j) \in A, \tag{5.141c}$$

$$x_{ij} \ge 0. \tag{5.141d}$$

The equations (5.141b) can be expanded as follows :

$$\sum_{(p,q)\in A(T^0)} x_{pq} = y_{ij} \quad \text{for every } p \approx (i, j) \in N_1(T^0). \tag{5.142a}$$

$$-\sum_{(p,q)\in A(T^0)} x_{pq} = y_{kl} \quad \text{for every } q \approx (k, l) \in N_2(T^0). \tag{5.142b}$$

Note that $N_1(T^0) = T^0$, $N_2(T^0) = A - T^0$ and $p \approx (i, j)$ means p is the node in $G(T^0)$ corresponding to the arc (i, j) in G.

Multiplying equations (5.142a) and (5.142b) by c_{ij} and c_{kl} respectively, we get

$$\sum_{(i,j)\in A} c_{ij}\, y_{ij} = \sum_{(p,q)\in A(T^0)} (c_{ij} - c_{kl})\, x_{pq}\,, \tag{5.143}$$

where, for an $(p, q) \in A(T)$, $p \approx (i, j) \in A\text{-}T^0$ and $q \approx (k, l) \in T^0$, it represents the relation $(i, j) \in [S_k, S_l]$.

Furthermore, since $x_{pq} \ge 0$ and $b_{ij} > 0$, the condition $y_{ij} \ge -b_{ij}$ is redundant for $(i, j) \in A\text{-}T^0 = N_1(T^0)$ as evident from the equation (5.142). Similarly the condition $y_{kl} \le b_{kl}$ is redundant for $(k, l) \in T^0 = N_2(T^0)$.

Let us define

$$\hat{c}_{pq} = c_{ij} - c_{kl} \quad \text{for every } (p, q) \in A(T^0) \tag{5.144}$$

where $p \approx (i, j) \in A\text{-}T^0$ and $q \approx (k, l) \in T^0$.

Incorporating the above discussion in the formulation of (DWRIST), we have the following result.

Theorem 5.21 : The dual of WRIST is the following transportation problem on the cut-graph $G(T^{\circ})$:

$$\text{- Minimize} \quad \sum_{(p,q)\in A(T^{\circ})} \hat{c}_{pq}\, x_{pq} \tag{5.145a}$$

subject to

$$\sum_{(p,q)\in A(T^{\circ})} x_{pq} \le a_{ij} \quad \text{for every } p \approx (i,\ j) \in A\text{-}T^{\circ}, \tag{5.145b}$$

$$\sum_{(p,q)\in A(T^{\circ})} x_{pq} \le b_{kl} \quad \text{for every } q \approx (k,\ l) \in T^{\circ}, \tag{5.145c}$$

$$x_{pq} \ge 0 \qquad \text{for every } (p,\ q) \in A(T^{\circ}), \tag{5.145d}$$

where $\hat{c}_{pq}$'s are defined by (5.144).

Special case : Rectilinear Variance

In this case, $a_{ij} = b_{ij} = 1$. Hence the formulation of the dual DWRIST of WRIST becomes the following assignment problem.

$$\text{- Minimize} \quad \sum_{(p,q)\in A(T^{\circ})} \hat{c}_{pq}\, x_{pq} \tag{5.146a}$$

subject to

$$\sum_{(p,q)\in A(T^{\circ})} x_{pq} \le 1 \quad \text{for every } p \approx (i,\ j) \in A\text{-}T^{\circ}, \tag{5.146b}$$

$$\sum_{(p,q)\in A(T^{\circ})} x_{pq} \le 1 \quad \text{for every } q \approx (k,\ l) \in T^{\circ}, \tag{5.146c}$$

$$x_{pq} \ge 0 \qquad \text{for every } (p,\ q) \in A(T^{\circ}), \tag{5.146d}$$

where $\hat{c}_{pq}$'s are defined by (5.144).

Chapter 6

CONCLUDING REMARKS

In this dissertation, we have studied three primal algorithms for the minimum cost flow problem and two types of cost perturbations.

The primal algorithms that we consider are variants of well-known primal algorithms, the network simplex algorithm and the cycle-cancelling algorithm. Our variant of the network simplex algorithm chooses an arc having minimum cost-to-penalty ratio as the entering arc in each iteration; the penalty of an arc is one if it violates optimality conditions in the initial basis, otherwise, zero. Our cycle-cancelling algorithm with capacity scaling augments repeatedly a flow along a negative cycle which reduces the residual capacity of an eligible arc with maximum residual capacity by a factor of at least two. Our cancel-and-share algorithm consists of repeatedly increasing the potentials by a sufficiently large amount on a subset of nodes and making the resulting admissible graph acyclic.

Our primal network simplex algorithm runs in $O(\Delta\ (m+n \log n)$ time, where Δ is an upper bound on the sum of arc flows. The algorithm achieves the best time complexity for the network

simplex algorithm for the assignment problem and the shortest path problem. The cycle-cancelling algorithm with capacity scaling and the enhanced cancel-and-share algorithm run respectively in $O(m \log U(m+n \log n)$ time and in $O(nm \log n \min\{\log(nC), on \log n\})$ time.

One investigation that can be taken for further research is obtaining a strongly polynomial time implementation of the cycle-cancelling algorithm with capacity-scaling. An open question posed by Goldberg and Tarjan is whether cancel subroutine can be implemented in $O(m \log(n^2/m))$ time. If this question can be settled positively, then Goldberg and Tarjan's cancel-and-tighten algorithm and the cancel-and-share algorithm, proposed in this dissertation, will give one of the best complexity bound obtained for the minimum cost flow problem. The empirical behaviour of our algorithm can also be studied.

We have also studied cost perturbation of an arc that preserves the optimality of a given solution. Our approach is applicable to both the minimum cost flow problem and the separable convex cost flow problem. All cost perturbations are characterized by arc tolerance intervals. Using reoptimization, we developed an $O(n^3)$ algorithm that finds nonsingleton shortest paths between all pairs of nodes, which directly give tolerance intervals for all arcs. Since our approach eliminates the effects of degeneracy, efficient implementations of our algorithm will be useful in practice.

In inverse network flow problems, we have studied the problems of finding a cost for which a given flow is optimal and whose variance is minimum. We have shown duals of these problems are the problems of finding certain normalized feasible (augmenting) flow directions. Duals of the inverse network flow problems with side constraints can be considered for further research.

REFERENCES

Ahuja, R.K., and J.B. Orlin. 1992. The Scaling network simplex algorithm. *Operations Research*, 40, Supplement 1, S5-S13.

Ahuja, R.K., A.V. Goldberg, J.B. Orlin, and R.E. Tarjan. 1992. Finding minimum cost flows by double scaling. *Mathematical Programming*, 53, 243-266.

Ahuja, R.K., T.L. Magnanti, and J.B. Orlin. 1989. Network Flows, In *Handbooks in Operations Research Management Science*, Vol. 1; *Optimizations*, edited by G.L. Nemhauser, A.H.G. Rinnooy Kan and M.J. Todd, North-Holland, Amsterdam, pp. 211-369.

Ahuja, R.K., T.L. Magnanti, and J.B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ.

Ahuja, R.K., T.L. Magnanti, J.B. Orlin, and M.R. Reddy. 1995. Applications of Network Optimization. In *Handbooks in Operations Research and Management Science*, Vol. 7: *Network Models*, edited by M.O. Ball.

Akgul, M. 1993. A genuinely polynomial primal simplex algorithm for the assignment problem. *Discrete Applied Mathematics*, 45, 93-155.

Akgul, M. 1985a. Shortest path and simplex method. Research Report, Department of Computer Science and Operations Research, North Carolina State University, Raleigh, NC.

Barahona, F., and E. Tardos. 1989. Note on Weintraub's minimum cost circulation algorithm. ***SIAM Journal on Computing***, 18, 579-583.

Barr, R.S., F. Glover, and D. Klingman. 1977. The alternating path basis algorithm for the assignment problem. ***Mathematical Programming***, 13, 1-13.

Bazaraa, M.S., J.J. Jarvis, and H.D. Sherali. 1990. ***Linear Programming and Network Flows***, 2nd ed., Wiley, New York.

Bennington, G.E. 1974. Applying network analysis, ***Industrial Engineering***, 6, 17-25.

Ben Israel, A., and S.D. Flåm. 1989. Support Prices of Activities in Linear Programming. ***Optimization***, 20(5), 561-579.

Bertsekas, D.P. 1979. A distributed algorithm for the assignment problem, Working Paper, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Bertsekas, D.P., and P. Tseng. 1988. Relaxation methods for minimum cost ordinary and generalized network flow problems. ***Operations Research***, 36, 93-114.

Bland, R.G., and D.L. Jensen. 1992. On the computational behavior of a polynomial-time network flow algorithm. ***Mathematical Programming***, 54, 1-39.

Bradley, S.P., A.C. Hax, and T.L. Magnanti. 1977. ***Applied Mathematical Programming***, Addison-Wesley, Reading, MA.

Burton, D., and Ph. L. Toint. 1992. On an instance of the inverse shortest path problem. ***Mathematical Programming***, 53, 45-61.

Burton, D., and Ph. L. Toint. 1994. On the use of an inverse shortest paths algorithm for recovering linearly correlated costs. *Mathematical Programming*, 63, 1-22.

Busakar, R.G., and P.J. Gowen. 1961. A procedure for determining minimal-cost network flow patterns. ORO Technical Report 15, Operational Research Office, John Hopkins University, Baltimore, MD.

Cormen, T.H., C.E. Leiserson, and R.L. Rivest. 1990. *Introduction to Algorithms*, MIT Press and McGraw-Hill, New York.

Cunningham, W.H. 1976. A network simplex method. *Mathematical Programming*, 11, 105-116.

Cunningham, W.H. 1979. Theoretical properties of the network simplex method. *Mathematics of Operations Research*, 4, 196-208.

Dantzig, G.B. 1951a. Maximization of a linear function of variables subject to linear inequalities. In *Activity Analysis of Production and Allocation*, edited by T.C. Koopmans, Wiley, New York, pp. 339-347.

Dantzig, G.B. 1951b. Application of the simplex method to a transportation problem. In *Activity Analysis and Production and Allocation*, edited by T.C. Koopmans, Wiley, New York, pp. 359-373.

Dantzig, G.B. 1962. *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ.

Edmonds, J., and R.M. Karp. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of ACM*, 19, 248-264.

Ervolina, T.R., and S.T. McCormick. 1990. Two strongly polynomial cut cancelling algorithms for minimum cost network flow. Technical Report, Faculty of Commerce and Business Administration, University of British Columbia, Vancouver, Canada.

Fong, C.O., and V. Srinivasan. 1977. Determining all nondegenerate shadow prices for the transportation problem, *Transportation Science*, 11(3), 199-222.

Ford, L.R., and D.R. Fulkerson. 1957. A primal-dual algorithm for the capacitated Hitchcock Problem. *Naval Research Logistics Quarterly*, 4, 47-54.

Ford, L.R., and D.R. Fulkerson. 1962. *Flows in Networks*, Princeton University Press, Princeton, NJ.

Fredman, M.L., and R.E. Tarjan. 1984. Fibonacci heaps and their uses in improved network optimization algorithms. Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science, pp. 338-346, Full paper in *Journal of ACM*, 34 (1987), 596-615.

Fujishige, S. 1986. A capacity-rounding algorithm for the minimum cost circulation problem: A dual framework of Tardos algorithm. *Mathematical Programming*, 35, 298-308.

Gabow, H.N., and R.E. Tarjan. 1989. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18, 1013-1036.

Gal, T. 1979. *Postoptimal Analysis, Parametric Programming and Related Topics*, McGraw-Hill, New York.

Gal, T. 1984. Linear parametric programming - A brief survey. *Mathematical Programming Study*, 21, 43-68.

Galil, Z., and É. Tardos. 1986. An $O(n^2(m+n \log n) \log n)$ min-cost flow algorithm. Proceedings of the 27th Annual Symposium on the Foundations of Computer Science, pp. 136-144, Full paper in *Journal of ACM*, 35 (1987), 374-386.

Gasner, B.J. 1964. Cycling in the transportation problem. *Naval Research Logistics Quarterly*, 11, 43-58.

Glover, F., D. Karney, D. Klingman, and A. Napier. 1974. A computational study on start procedures, basis change criteria, and solution algorithms for transportation problem. *Management Science*, 20, 793-813.

Glover, F., and D. Klingman. 1977. Network applications in industry and government. *AIIE Transactions*, 9, 363-376.

Glover, F., and D. Klingman. 1988. A pseudopolynomial time primal simplex algorithm for minimum cost network flow problem. Paper presented at the International TIMS Conference, Paris, July 1988, and the 13th International Symposium on Mathematical Programming, Tokyo, August 1988.

Glover, F., D. Klingman, and N. Phillips. 1990. Netform modeling and applications. *Interfaces*, 20, 7-27.

Goldberg, A.V., E. Tardos, and R.E. Tarjan. 1989. Network flow algorithms, Technical Report 860, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY.

Goldberg, A.V., and R.E. Tarjan. 1987. Solving minimum cost flow problem by successive approximation. Proceedings of the 19th ACM Symposium on the Theory of Computing, pp. 7-18, Full Paper in *Journal of ACM*, 35 (1988), 921-940.

Goldberg, A.V., and R.E. Tarjan. 1988. Finding minimum-cost circulations by cancelling negative cycles. Proceedings of the

20th ACM Symposium on the Theory of Computing, pp. 388-397. Full Paper in *Journal of ACM*, 36 (1989), 873-886.

Goldberg, A.V., M.D. Grigoriadis, and R.E. Tarjan. 1991. Efficiency of the network simplex algorithm for the maximum flow problem. *Mathematical Programming*, 50, 277-296.

Golden, B.L., and T.L. Magnanti. 1977. Deterministic network optimization: A bibliography, *Networks*, 7, 149-183.

Goldfarb, D., and A. Idnani. 1983. A numerically stable dual method for solving strictly convex quadratic problems, *Mathematical Programming*, 27, 1-33.

Goldfarb, D., and J. Hao. 1988. Polynomial-time primal simplex algorithms for the minimum cost network flow problem. Technical Report, Department of Industrial Engineering and Operations Research, Columbia University, NY.

Goldfarb, D., and J. Hao. 1990. A primal simplex algorithm that solves the maximum flow problems in at most *nm* pivots and $O(n^2m)$ time. *Mathematical Programming*, 47, 353-365.

Goldfarb, D., J. Hao, and S. Kai. 1990a. Efficient shortest path simplex algorithms. *Operations Research*, 38, 624-628.

Goldfard, D., J. Hao, and S, Kai. 1990b. Anti-stalling pivot rules for the network simplex algorithm. *Networks*, 20, 79-91.

Gupta, S.K. 1985. *Linear Programming and Network Models*, Affiliated East-West Press, New Delhi, India.

Gusfield, D. 1983. A note on arc tolerances in sparse shortest-path and network flow problems. *Networks*, 13, 191-196.

Hausman, H. 1978. Integer Programming and Related Areas : A Classified Bibliography, Lecture Notes in Economics and Mathematical Systems, Vol. 160, Springer-Verlag, Berlin.

Hitchcock, F.L. 1941. The distribution of a product from several sources to numerous facilities. *Journal of Mathematical Physics*, 20, 224-230.

Hung, M.S. 1983. A polynomial simplex method for the assignment problem. *Operations Research*, 31, 595-600.

Iri, M. 1960. A new method of solving transportation-network problems. *Journal of Operations Research Society of Japan*, 3, 27-87.

Jewell, W.S. 1958. Optimal flow through networks. Interim Technical Report 8, Operations Research Center, MIT, Cambridge, MA.

Johnson, E.L. 1966. Networks and basic solutions. *Operations Research*, 14, 619-624.

Kantorovich, L.V. 1939. Mathematical methods in organization and planning of production, Publication House of the Leningrad University, Translated in *Management Science*, 6, (1960), 366-422.

Karp, R.M., and J.B. Orlin. 1981. Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Applied Mathematics*, 3, 37-45.

Kastning, C. 1976. Integer Programming and Related Areas : A Classified bibliography. Lecture Notes in *Economics and Mathematical Systems*, Vol. 128, Springer-Verlag, Berlin.

Kennington, J.L., and R.V. Helgason. 1980. *Algorithms for Network Programming*, Wiley-Interscience, New York.

Klein, M. 1967. A primal method for minimal cost flows with application to the assignment and transportation problems. *Management Science*, 14, 205-220.

Koopmans, T.C. 1947. Optimum utilization of the transportation system, Proceedings of the International Statistical Conference, Washington, DC, Also in Econometrica, 17 (1949).

Lawler, E.L. 1976. *Combinatorial Optimization: Networks and Matroids*, Holt. Rinehart and Winston, New York.

Minieka, E. 1978. *Optimization Algorithms for Networks and Graphs*, Marcel Dekker, New York.

Mulvey, J. 1978. Pivot strategies for primal-simplex network codes. *Journal of ACM*, 25, 266-270.

Murty, K.G. 1976. *Linear and Combinatorial Programming*, John Wiley & Sons, New York.

Orlin, J.B. 1984. Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem. Technical Report, 1615-84, Sloan School of Management, MIT, Cambridge, MA.

Orlin, J.B. 1985. On the simplex algorithm for networks and generalized networks. *Mathematical Programming Study*, 24, 166-178.

Orlin, J.B. 1988. A faster strongly polynomial minimum cost flow algorithm, Proceedings of the 20th ACM Symposium on the Theory of Computing, pp. 377-387. Full paper to appear in *Operations Research*.

Orlin, J.B. 1995. A polynomial time primal network simplex algorithm for minimum cost flows, Technical Report, Sloan School of Management, MIT, Cambridge, MA.

Srinivasan, V., and G.L. Thompson. 1972. An operator theory of parametric programming for the transportation problem. *Naval Research Logistic Quarterly*, 19, 205-252.

Srinivasan, V., and G.L. Thompson. 1973. Benefit-cost analysis of coding techniques for primal transportation algorithm. *Journal of ACM*, 20, 194-213.

Srinivasan, V., and G.L. Thompson. 1977. Cost operator algorithms for the transportation problem. *Mathematical Programming*, 12, 372-391.

Tardos, É. 1985. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5, 247-255.

Tarjan, R.E. 1982. Sensitivity analysis of minimum spanning trees and shortest path trees. *Information Processing Letters*, 14, 30-33.

Tarjan, R.E. 1991. Efficiency of the primal network simplex algorithm for the minimum-cost circulation problem. *Mathematics of Operations Research*, 16, 272-291.

Von Randow, R. 1985. *Integer Programming and Related Areas: A Classified Bibliography 1981-1984*, Lecture Notes in Economics and Mathematical Systems, Vol. 197, Springer- Verlag, Berlin.

Wallacher, C. 1991. A generalization of the minimum-mean cycle solution rule in cycle cancelling algorithms, Technical Report, Inst. fur Ang. math., Braunschweig.

Wallacher, C., and U.T. Zimmermann. 1991. A combinatorial interior point method for network flow problems, Presented at the *14th International Symposium on Mathematical Programming*, Amsterdam, The Netherlands.

Wendell, R.E. 1985. The tolerance approach to sensitivity analysis in linear programming. ***Management Science***, 31(5), 564-578.

Young, N.E., R.E. Tarjan, and J.B. Orlin. 1991. Faster parametric shortest path and minimum-balance algorithms. ***Networks***, 21, 205-221.

Iri, M. 1969. *Network Flow, Transportation and scheduling.* Academic Press, New York.